Available at
www.ElsevierComputerScience.com
POWERED BY SCIENCE DIRECT®

Information and Computation 187 (2003) 291–319

Information and Computation

www.elsevier.com/locate/ic

Describing parameterized complexity classes

Jörg Flum[a] and Martin Grohe[b,*]

[a] *Abteilung für Mathematische Logik, Albert-Ludwigs-Universität Freiburg, Eckerstr. 1, Freiburg 79104, Germany*
[b] *Institut für Informatik, Humboldt-Universität zu Berlin, Unter den Linden 6, 10099 Berlin, Germany*

Received 2 April 2002; revised 6 June 2003

Abstract

We describe parameterized complexity classes by means of classical complexity theory and descriptive complexity theory. For every classical complexity class we introduce a parameterized analogue in a natural way. In particular, the analogue of polynomial time is the class of all fixed-parameter tractable problems. We develop a basic complexity theory for the parameterized analogues of classical complexity classes and give, among other things, complete problems and logical descriptions. We then show that most of the well-known intractable parameterized complexity classes are not analogues of classical classes. Nevertheless, for all these classes we can provide natural logical descriptions.

1. Introduction

Parameterized complexity theory provides a framework for a refined complexity analysis, which in recent years has found its way into various areas of computer science, such as database theory [19,24], artificial intelligence [17], and computational biology [3,26]. Central to the theory is the notion of *fixed-parameter tractability*, which relaxes the classical notion of tractability, polynomial time computability, by admitting algorithms whose runtime is exponential, but only in terms of some *parameter* that is usually expected to be small. As a complexity theoretic counterpart, a theory of *parameterized intractability* has been developed. Remarkably, complexity theoretic assumptions from this theory of parameterized intractability have recently been used to prove results in a purely classical setting [2,20], which gives some significance to the theory beyond its obvious purpose of showing that certain problems are not fixed-parameter tractable.

* Corresponding author.
E-mail addresses: flum@uni-freiburg.de (J. Flum), grohe@informatik.hu-berlin.de (M. Grohe).

0890-5401/$ - see front matter © 2003 Elsevier Inc. All rights reserved.
doi:10.1016/S0890-5401(03)00161-5

Unfortunately, the landscape of fixed-parameter intractable problems is not as nice and simple as the landscape of classical intractable problems provided by the theory of NP-completeness. Instead, there is a huge variety of seemingly different classes of fixed-parameter intractable problems. Moreover, the definitions of these classes are quite intractable themselves, at least for somebody who is not an expert in the area. One reason for this is that the classes are not defined in terms of a simple machine model, but as the closure of particular problems under parameterized reductions. Indeed, for the most interesting of these classes there seem to be no nice descriptions in terms of, say, Turing machines.[1] However, in this paper we give simple and uniform logical descriptions of the classes using variants of logics that are also used in classical descriptive complexity theory. A related issue we address here is the relation between parameterized and classical complexity classes. In particular when proving classical results under assumptions from parameterized complexity theory, an understanding of these assumptions in classical terms would be desirable. We develop a theory of parameterized complexity classes derived from classical classes and set it in the context of the existing structural parameterized complexity theory.

1.1. Fixed-parameter tractability

An instance of a *parameterized problem* is a pair (x, k) consisting of the actual input x and a parameter k.[2] Thus formally a parameterized problem is a subset of $\Sigma^* \times \mathbb{N}$ for some finite alphabet Σ. Often, parameterized problems P are derived from classical problems $X \subseteq \Sigma^*$ by choosing a suitable parameterization $p : \Sigma^* \to \mathbb{N}$ and letting $P = \{(x, p(x)) \mid x \in X\}$. The idea is to choose the parameterization in such a way that for those instances that most often occur in practice the parameter value is small. An illustrative example is the problem of evaluating a database query. Its input consists of a relational database and a query, and a reasonable parameterization of this problem is by the length of the input query, which can be expected to be small compared to the size of the input database. It is important, however, to keep in mind that every classical problem has different parameterizations that may lead to parameterized problems of drastically different (parameterized) complexities. Choosing a natural and useful parameterization is part of the problem analysis and modelling, and it may very well be the case that in different situations different parameterizations of the same problem are appropriate.

Parameterized complexity theory is based on a more relaxed notion of tractability in which one considers a runtime of an algorithm for a parameterized problem that is only exponential in terms of the (presumably small) parameter as tractable. Let $P \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. We always denote the parameter by k and the length $|x|$ of the input string x by n. The problem P is *fixed-parameter tractable* if there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c, and an algorithm deciding P in time $O(f(k) \cdot n^c)$. Our starting point is the observation that this notion of fixed-parameter tractability is very robust and has other natural interpretations. The first of these is inspired by the database query evaluation example. Instead of requiring a query to be evaluated in polynomial time, we may allow an algorithm to first optimise the input query, that is, to transform it to an equivalent query that is easier to evaluate, and then evaluate the optimised query in polynomial time (see [19]). In our abstract setting, we allow

[1] Note added in proof: in the meantime, machine-characterisations using random access machines have been obtained for many of these classes [6,7].

[2] It is well known that for the general theory it represents no restriction to assume that the parameters are natural numbers.

an algorithm to first do a pre-computation that only involves the parameter and then use the output of the pre-computation and the original input to decide the instance in polynomial time. Formally, we say that a parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ is in *polynomial time after a pre-computation* if there is an algorithm $\mathbb{P}$ that, given a natural number k, produces some string $\mathbb{P}(k)$ and there is a polynomial time algorithm $\mathbb{A}$ that for an instance (x, k) of P gets the input $(x, \mathbb{P}(k))$ and decides whether $(x, k) \in P$. Then P is fixed-parameter tractable if, and only if, it is in polynomial time after a pre-computation. The following characterisation of fixed-parameter tractability is more technical, but it contributed a lot to our intuitive understanding of fixed-parameter tractability: we say that $P \subseteq \Sigma^* \times \mathbb{N}$ is *eventually in polynomial time* if there is a computable function f and a polynomial time algorithm that, given an instance (x, k) of P with $n \geqslant f(k)$, decides if $(x, k) \in P$. Then P is fixed-parameter tractable if, and only if, it is computable and eventually in polynomial time.

1.2. Parameterized classes derived from classical classes

If the class FPT is a parameterized version of PTIME, can we understand other parameterized complexity classes in a similar way as being derived from classical complexity classes? This is the first problem we shall study in this paper. For every complexity class K we introduce a parameterized complexity class para-K, taking FPT = para-PTIME as a prototype. Our mechanism of deriving a parameterized complexity class para-K from a classical class K is closely related to a similar mechanism introduced by Cai et al. [5]. We show that the classes para-K are quite robust; for all reasonable complexity classes K, the class para-K can equivalently be characterised as the class of all problems being in K after a pre-computation, the class of all computable problems being eventually in K, and by a resource bound similar to that in the original definition of fixed-parameter tractability. After we have thus seen that the classes para-K are quite natural, we develop a basic complexity theory of these classes. It turns out that the classes para-K are closely related to the corresponding K: para-K is contained in para-K′ if, and only if, K is contained in K′. Complete problems for the parameterized classes para-K under parameterized reductions are essentially unparameterized problems that are also complete for the corresponding K under standard reductions. Moreover, the classes para-K admit logical descriptions closely related to such descriptions of the corresponding K.

Overall, we get a fairly accurate picture of the structure of these parameterized complexity classes; it simply reflects the structure of the corresponding classical classes.

1.3. The W-hierarchy and other "inherently parametric" classes

The most important parameterized complexity classes of (apparently) intractable parameterized problems are the classes of Downey and Fellow's W-hierarchy [8]. Other classes we shall study here are AW[$*$] [1] and the classes of the A-hierarchy [15]. Many natural parameterized problems that are not known to be fixed-parameter tractable have been shown to be complete for one of these classes, most of them for the first level W[1] or second level W[2] of the W-hierarchy. Therefore, the main goal of our research in structural parameterized complexity theory is a better understanding of these classes. Remembering the previous paragraph, we may hope that the intractable parameterized complexity classes are derived from intractable classical classes such as NP and the classes of the polynomial hierarchy. Unfortunately, the situation is not so simple; we show that none of the classes Q mentioned above is of the form para-K for a classical class K (unless Q = FPT). The classes turn out to be interwoven with

para-NP, the parameterized classes para-Σ_t^{P} derived from the levels of the polynomial hierarchy, and para-PSPACE in a non-trivial way.

So we cannot really use our structure theory for the classes para-K to understand the classes of W-hierarchy, the A-hierarchy, and AW[$*$]. However, we can extend our descriptive complexity theoretic approach and give logical descriptions of all these classes in terms of finite-variable fixed-point logics. For example, the class W[t], the tth level of the W-hierarchy, is defined to be the class of all problems that are reducible to a parameterized version of the satisfiability problem for a certain class of Boolean circuits. Our characterisation says that W[t] is the class of all problems *slicewise definable* in a certain fragment of fixed-point logic. Descriptive characterisations of classical complexity classes have been criticised for being merely simple translations of the original Turing machine based definitions of the classes into a logical formalism and therefore not providing new insights. Although we do not entirely agree with this criticism, it is certainly true that the logical characterisations are often very close to the machine descriptions. However, in our context this is rather an advantage, because there are no known simple and natural machine descriptions of the parameterized classes we characterise here.

1.4. Organisation and preliminaries

We have organised the paper following the outline given in this introduction. We assume that the reader is familiar with the fundamentals of complexity theory (see, e.g. [25]) and logic (see, e.g. [13]). We do not assume familiarity with parameterized complexity theory. For background in this area, we refer to [9].

2. Parameterized classes derived from classical classes

Classical problems, which we usually denote by X, Y, are subsets of Σ^* for some alphabet Σ. *Parameterized problems*, which we denote by P, Q, are subsets of $\Sigma^* \times \mathbb{N}$. Of course, we may view any parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ as a classical problem in some larger alphabet, say, the alphabet obtained from Σ by adding new symbols '(', ',', ')', and '1.'[3] Conversely, with every classical problem $X \subseteq \Sigma^*$ we can associate its *trivial parameterization* $X \times \{0\}$. A classical or parameterized complexity class is simply a set of classical problems or parameterized problems, respectively.

In this section, we associate a parameterized complexity class para-K with every classical complexity class K and then discuss basic properties of these classes and their relation to the classical counterparts.

Definition 1. Let K be a classical complexity class. Then *para-K* is the class of all parameterized problems P, say with $P \subseteq \Sigma^* \times \mathbb{N}$, for which there exists an alphabet Π, a computable function $f : \mathbb{N} \to \Pi^*$, and a problem $X \subseteq \Sigma^* \times \Pi^*$ such that $X \in \mathrm{K}$ and for all instances $(x, k) \in \Sigma^* \times \mathbb{N}$ of P we have

$$(x, k) \in P \iff (x, f(k)) \in X.$$

Intuitively, para-K consists of all problems that are in K *after a pre-computation* that only involves the parameter. This means that a problem in para-K can be solved by two algorithms $\mathbb{P}$ and $\mathbb{A}$, where $\mathbb{P}$

[3] Parameters are usually encoded in unary, although this is inessential.

is arbitrary and $\mathbb{A}$ has resources constrained by K. The pre-computation $\mathbb{P}$ only involves the parameter; it transforms k into some string $f(k)$, presumably one that makes it easier to solve the problem. Then the algorithm $\mathbb{A}$ solves the problem, given the original input x and the result $f(k)$ of the pre-computation, with resources constrained by the complexity class K.

Example 2. We give a typical example of a problem solution involving a pre-computation. We take Büchi's [4] well-known characterisation of regular languages by sentences of monadic second-order logic (MSO) to evaluate MSO-sentences on strings. The input of this problem consists of a string x and a sentence φ of monadic second-order logic. We assume that we have fixed a reasonable encoding $[\cdot]$ of MSO-sentences by natural numbers. We parameterize the problem by the encoding $[\varphi] \in \mathbb{N}$ of the input sentence φ. The pre-computation translates $[\varphi]$ into an equivalent finite automaton $\mathfrak{A}(\varphi)$. Then the main algorithm runs $\mathfrak{A}(\varphi)$ on the string x; this only requires polynomial time.[4]

We have just proved that the problem of evaluating a sentence of monadic second-order logic on a string, parameterized by an encoding of the input sentence, is in FPT = para-PTIME.[5]

Our definition of para-K is almost the same as the definition of the class *uniform K + advice*, which Cai et al. [5] introduced as a parameterized version of the complexity class K. Instead of considering the result $f(k)$ of the pre-computation as part of the input of the main computation, Cai et al. treat it as an "advice string," which is accessed as an "oracle." Our model is slightly more general, because by treating it as part of the input we do not have to worry about how the result of the pre-computation is accessed, and we have to make no reference to a particular machine model. But of course the two notions are equivalent for all natural complexity classes, in particular for all those called "regular" below.

We have mentioned in Section 1 that para-PTIME is precisely the class FPT of all fixed-parameter tractable problems (formally, this is an immediate consequence of Theorem 4 below and also of the results of [5]). It seems somewhat arbitrary to use the pre-computation view on fixed-parameter tractability as the basis of our generalisation of FPT = para-PTIME to other classes K. However, it turns out that the equivalent characterisations of FPT given in the introduction have an analogue for any class para-K derived from a "reasonable" complexity class K. In defining which complexity classes we consider reasonable, we are not trying to be as general and abstract as possible, but we want to have a notion that includes all the standard classes such as LOGSPACE, NLOGSPACE, PTIME, NP, Σ_i^{P} (the ith level of the polynomial hierarchy) for $i \geqslant 1$, PSPACE, etc. We first fix a machine model $\mathcal{M}$, for our purposes it seems best to use alternating Turing machines with some restriction put on the alternations that are allowed. Our Turing machines have an input tape, if necessary an output tape, and an arbitrary finite number of work tapes. For example, $\mathcal{M}$ may just be the class of deterministic Turing machines, or the class of non-deterministic Turing machines, or maybe the class of alternating Turing machines with at most five quantifier alternations starting with an existential state. Then we fix a resource $\mathcal{R}$, either time or space. For a function $\gamma : \mathbb{N} \to \mathbb{N}$, we let $\mathrm{K}(\mathcal{M}, \mathcal{R}, \gamma)$ be the class of all problems accepted by a

[4] In fact, it only requires linear time, if as underlying machine model we take random access machines with a uniform cost measure. Then our argument shows that the problem is in para-LINTIME.

[5] Let us remark that the same is true for the more natural parameterization of the problem by the length of the input formula. In fact, it is easy to see that the two parameterizations of the problem (by an encoding of the input formula and the length of the input formula, respectively) yield parameterized problems that are equivalent with respect to suitable parameterized reductions.

machine in $\mathcal{M}$ that on all inputs of size n uses an amount of at most $\gamma(n)$ of $\mathcal{R}$, for all $n \in \mathbb{N}$. For a class Γ of functions we let $\mathrm{K}(\mathcal{M}, \mathcal{R}, \Gamma) = \bigcup_{\gamma\in\Gamma} \mathrm{K}(\mathcal{M}, \mathcal{R}, \gamma)$.

We call a class Γ of functions on the natural numbers *regular*, if for all $\gamma \in \Gamma$ we have:

- γ is computable.
- γ is monotone growing with $\lim_{m\to\infty} \gamma(m) = \infty$.
- For all $c \in \mathbb{N}$ there are functions $\gamma_1, \gamma_2, \gamma_3 \in \Gamma$ such that for all $m \in \mathbb{N}$ we have $c + \gamma(m) \leqslant \gamma_1(m)$, $c \cdot \gamma(m) \leqslant \gamma_2(m)$, and $\gamma(c \cdot m) \leqslant \gamma_3(m)$.

A *regular complexity class* is a class $\mathrm{K}(\mathcal{M}, \mathcal{R}, \Gamma)$ for a regular class Γ of functions. It is easy to see that all the complexity classes mentioned above are regular.

Recall that n always denotes the length of the input x of a parameterized problem and k denotes the parameter.

Definition 3. Let $P \in \Sigma^* \times \mathbb{N}$ be a parameterized problem and K a classical complexity class. Then P is *eventually in* K if there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a problem $X \in \mathrm{K}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ with $n \geqslant f(k)$ we have

$$(x, k) \in P \iff (x, k) \in X.$$

Theorem 4. *Let* $\mathrm{K} = \mathrm{K}(\mathcal{M}, \mathcal{R}, \Gamma)$ *be a regular complexity class. Then for every parameterized problem* $P \subseteq \Sigma^* \times \mathbb{N}$ *the following three statements are equivalent*:

(1) P *is in* para-K.
(2) P *is computable and eventually in* K.
(3) *There is a computable function* f *and a function* $\gamma \in \Gamma$ *such that* $P \in \mathrm{K}(\mathcal{M}, \mathcal{R}, f(k) + \gamma(n))$.

Proof. In the following proof, we assume that the parameters are encoded in unary. This slightly simplifies the arguments. The results would remain true if we would use a binary encoding.

To prove (1) $\Rightarrow$ (3), suppose that $P \subseteq \Sigma^* \times \mathbb{N}$ is in para-K, and that this is witnessed by the computable function $f : \mathbb{N} \to \Pi^*$, for some alphabet Π, and the problem $X \subseteq \Sigma^* \times \Pi^*$ in K. Let $M_X \in \mathcal{M}$ be a Turing machine that decides X with resource $\mathcal{R}$ bounded by the function $\gamma \in \Gamma$. Let M_f be a deterministic Turing machine computing f. We construct a computable function $h : \mathbb{N} \to \mathbb{N}$ and a machine $M \in \mathcal{M}$ deciding P whose $\mathcal{R}$-complexity is bounded by $h(k) + \gamma(2 \cdot n)$. By the regularity of Γ, there is a function $\gamma' \in \Gamma$ such that $h(k) + \gamma(2 \cdot n) \leqslant h(k) + \gamma'(n)$, so our machine M witnesses (3).

On input (x, k), machine M first simulates M_f to produce $f(k)$ on some worktape T. Then M simulates M_X on input $(x, f(k))$ on a set of additional worktapes. To be able to do this simulation, we have to modify the input behaviour of M_X in an appropriate way so that it can read x from the input tape and $f(k)$ from worktape T, instead of reading the pair $(x, f(k))$ from the input tape, but this can easily be done by modifying the transition table of M_X and adding states to simulate the correct behaviour at the ends of the strings. It neither requires additional time nor space. Let $g' : \mathbb{N} \to \mathbb{N}$ be the $\mathcal{R}$-complexity of M_f. Let $g : \mathbb{N} \to \mathbb{N}$ be defined by $g(m) = \max\{|f(m)|, g'(m)\}$; clearly g is computable. (If $\mathcal{R}$ is time, of course we have $g'(m) \geqslant f(m)$ for all m and thus $g = g'$ but this is not clear for space because the machine uses an output tape.) Then our machine M uses resources

$$g(k) + \gamma(|(x, f(k))|) = g(k) + \gamma(n + f(k) + c)$$

for some constant c. We let $h(k) := g(k) + \gamma(2(f(k) + c))$. Then if $n \leqslant f(k) + c$ we have $g(k)+$

$\gamma(n + f(k) + c) \leqslant h(k)$ (since γ is monotone).[6] If $n > f(k) + c$, then $g(k) + \gamma(n + f(k) + c) \leqslant h(k) + \gamma(2 \cdot n)$.

To prove (3) $\Rightarrow$ (2), let $P \in \mathrm{K}(\mathcal{M}, \mathcal{R}, f(k) + \gamma(n))$, for some computable function f and a function $\gamma \in \Gamma$. Without loss of generality we can assume that $f(k) \geqslant 2k + 2$. Let $M_P \in \mathcal{M}$ be a Turing machine that decides P with resource $\mathcal{R}$ bounded by the function $f(k) + \gamma(n)$. Let M_f be a deterministic Turing machine computing f. Let $g' : \mathbb{N} \to \mathbb{N}$ be the $\mathcal{R}$-complexity of M_f, and let $g : \mathbb{N} \to \mathbb{N}$ be defined by $g(m) = \max\{f(m), g'(m)\}$.

We construct a machine M as follows: on input (x, k), M simulates M_P, until the end of string x is reached. If this never happens, M just behaves like M_P. If the end of x is reached, M uses additional worktapes to test if $n < g(k)$. If this is the case, M rejects. Otherwise, M continues to simulate M_P. Thus if $n \geqslant g(k)$, M correctly decides P.

Let us analyse the $\mathcal{R}$-complexity of $\mathcal{M}$. We only give the argument for $\mathcal{R}$ being time, the argument for space is similar. There are two cases. If M_P never reads the whole input string x, than it behaves in exactly the same way as it would on input $(x, 1)$ instead of (x, k). Thus it needs at most $f(1) + \gamma(n)$ steps. Since in this case M behaves exactly like M_P, it also needs $f(1) + \gamma(n)$ steps. If M_P reads the whole string x, it needs at least n steps. In this case, M simulates at most all steps of M_P and in addition it needs at most $2(n + k + 1)$ steps to find out whether $g(k) \leqslant n$. To check whether $g(k) \leqslant n$, M proceeds as follows: when the subroutine starts, the input head is on the symbol separating x and k. M first copies k to a new worktape and moves the input head back to the symbol separating x and k; this requires $2(k + 1)$ steps. Then it simulates M_f, but in each step it moves the input head once cell to the left. If the input head of M reaches the left end of the input tape before M_f is finished, M just stops and rejects. Since by the definition of g, M_f runs for $g(k)$ steps, in this case we must have $n < g(k)$. Otherwise M_f finishes in at most n steps. In this case, M moves the input head back to the symbol separating x and k and continues the simulation. Thus the overall runtime of M in this case is bounded by $f(k) + \gamma(n) + 2(n + k + 1)$. Since we have $2(k + 1) \leqslant f(k) \leqslant g(k) \leqslant n$ and $n \leqslant \gamma(n)$, this term is at most $5\gamma(n)$.

Taking both cases together, we see that the runtime of M is bounded by $f(1) + 5\gamma(n)$, which is bounded by a function in Γ by regularity.

It remains to prove (2) $\Rightarrow$ (1). Suppose that $P \subseteq \Sigma^* \times \mathbb{N}$ is computable and eventually in K and that this is witnessed by the computable function $f : \mathbb{N} \to \mathbb{N}$ and the problem $X \subseteq \Sigma^* \times \mathbb{N}$ in K. Let $M_X \in \mathcal{M}$ be a Turing machine that decides X with resource $\mathcal{R}$ bounded by the function $\gamma \in \Gamma$.

There is a slight problem with the case that $\mathcal{R}$ is time and γ is sublinear. We shall discuss this problem later, for now let us assume that either $\mathcal{R}$ is space or $\mathcal{R}$ is time and $\gamma(m) \geqslant m$ for all $m \geqslant 1$. In both of these cases, we have enough resources to read the whole input tape, so if the input is a pair (x, y), we can skip over x and look at y first.

We shall define a computable function $g : \mathbb{N} \to \Pi^*$, for some alphabet Π, and a Turing machine $M \in \mathcal{M}$ with $\mathcal{R}$-complexity in Γ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ we have

$$(x, k) \in P \iff M \text{ accepts } (x, g(k)). \tag{1}$$

Let χ_P be the characteristic function of P. We first define a function g' by letting

$$g'(k) := \left(k, f(k), (x, \chi_P(x, k))_{\substack{x \in \Sigma^* \\ |x| < f(k)}}\right).$$

[6] Similar monotonicity arguments will be used several times implicitly in the following proof.

Thus essentially $g'(k)$ consists of k, $f(k)$, and a table storing whether $(x,k) \in P$ for all $x \in \Sigma^*$ with $|x| < f(k)$. Since P and f are computable, g' is computable. Let $h : \mathbb{N} \to \mathbb{N}$ be a computable function with the property that $\gamma(h(k)) \geqslant |g'(k)|$ and $h(k) \geqslant |g'(k)|^2$ for all $k \in \mathbb{N}$. Such an h exists because γ and g' are computable and $\lim_{m\to\infty} \gamma(m) = \infty$. Now we define $g : \mathbb{N} \to \Pi^*$ for a suitable $\Pi \supseteq \Sigma$ by letting $g(k)$ be the string $g'(k)$ padded to length $h(k)$ by adding '1's in the end, that is,

$$g(k) = g'(k)\ \underbrace{1\ldots1}_{h(k)-|g'(k)| \text{ times}}.$$

On input $(x, g(k))$, our machine M works as follows: it first compares $f(k)$ and $n = |x|$. Recall that $f(k)$ occurs as part of the string $g(k)$, so we do not have to compute it again. If $n < f(k)$, machine M looks up the the value $\chi_P(x,k)$ in the table $g'(k)$ and accepts or rejects, depending on this value. Otherwise, M simulates M_X.

Clearly, M satisfies (1). We have to prove that it also obeys the required resource bounds. Let m denote the length of the input $(x, g(k))$ and observe that $m = n + h(k) + c$ for some constant c. Let us first assume that $\mathcal{R}$ is time. Clearly the comparison between n and $f(k)$ only requires time $O(n + k) \subseteq O(n + h(k))$ (the comparison requires time $O(n)$; k additional steps are needed to move the input head over k, which appears between x and $f(k)$ in the input string). If $n < f(k)$, we can find the entry $\chi_P(x,k)$ in $g'(k)$ in time $O(n \cdot |g'(k)|) \subseteq O(f(k) \cdot |g'(k)|) \subseteq O(h(k))$. If $n \geqslant f(k)$, we can write (x,k) on a worktape, which requires time $O(n)$, and then simulate M_X in time $\gamma(|(x,k)|)$. The overall time we need is

$$d_1 \cdot n + d_2 \cdot h(k) + \gamma(|(x,k)|) \leqslant dm + \gamma(m) \leqslant (d+1)\gamma(m)$$

for suitable constants d_1, d_2, d. The last inequality holds by our assumption that $\gamma(m) \geqslant m$.

If $\mathcal{R}$ is space, we have to argue in a different way. To compare $f(k)$ and n, we copy $f(k)$ on a worktape and then do the comparison. This requires space $f(k) \leqslant |g'(k)| \leqslant \gamma(h(k)) \leqslant \gamma(m)$. Finding $\chi_P(n,k)$ in $g'(k)$ if $n < f(k)$ requires space $n < f(k) \leqslant \gamma(m)$, of course we can use the same space as before. Simulating M_X requires space $\gamma(n + k + c) \leqslant \gamma(m)$, so our machine M has space complexity at most $\gamma(m)$.

We still have to handle the case that $\mathcal{R}$ is time and $\gamma(m_0) < m_0$ for some $m_0 \in \mathbb{N}$. If $\gamma(m) \geqslant m - 2$ for all m, we can simply work with the function $2 + \gamma$ instead of γ, which is bounded by a function in Γ by regularity. So we can assume that $\gamma(m_0) \leqslant m_0 - 3$. This means that on all inputs (x,k) of length m_0, the machine M_X reads at most the first $m_0 - 3$ input symbols. In particular, this is the case for all inputs of length m_0 of the form $(x, 0)$. For such inputs, M_X can at most read the '$(x$' part of the input, and it never knows if it has read the full string x. M comes up with the correct answer by only reading the first $m_0 - 4$ symbols of x. This answer must also be correct for longer inputs, so M_X can actually be simulated by a Turing machine that works in constant time and space. Thus P can also be recognised by such a machine. □

Remark 5. For certain complexity classes $K = K(\mathcal{M}, \mathcal{R}, \Gamma)$, in particular for PTIME and PSPACE, Cai et al. [5] have proved that for every parameterized problem P, P is in para-K if, and only if, there is a computable function f and a function $\gamma \in \Gamma$ such that $P \in K(\mathcal{M}, \mathcal{R}, f(k) \cdot \gamma(n))$.

However, there are interesting classes such as LOGSPACE for which this equivalence does not seem to hold.[7]

[7] In the meantime, it has been shown that the equality fails for LOGSPACE [7].

Example 6. Besides FPT = para-PTIME, one of the most interesting classes of the form para-K seems to be para-LOGSPACE. By Theorem 4, a parameterized problem P is in para-LOGSPACE if, and only if, there is a computable function $f : \mathbb{N} \to \mathbb{N}$, a constant c, and an algorithm solving P in space $f(k) + c \cdot \log n$. Cai et al. [5] have observed that the natural parameterization of the vertex cover problem (by the size of the vertex cover) is in para-LOGSPACE. In this example we shall prove that many interesting properties of graphs of bounded degree are in para-LOGSPACE.

For a class D of structures and a class Φ of sentences of some logic, the *Φ-model-checking problem on D*, denoted by $\mathrm{MC}(D, \Phi)$, is the problem of deciding whether a given structure $\mathcal{A} \in D$ satisfies a given sentence $\varphi \in \Phi$. This problem is naturally parameterized by the length of the input sentence. For technical reasons that will become apparent later in this paper, we choose a slightly different parameterization (which can easily be seen to have the same parameterized complexity). We fix some encoding $[\cdot] : \Phi \to \mathbb{N}$ and parameterize the model-checking problem by $[\varphi]$. Formally, we let

$$\text{p-MC}(D, \Phi) = \{(\mathcal{A}, [\varphi]) \mid \mathcal{A} \in D, \varphi \in \Phi\}.$$

Denoting the class of all first-order sentences by FO and, for any $g \geqslant 1$, the class of all graphs of degree at most g by $\mathrm{DEG}(g)$, we claim:

$$\text{p-MC}(\mathrm{DEG}(g), \mathrm{FO}) \in \text{para-LOGSPACE}. \tag{2}$$

This implies that the standard parameterized versions of problems like dominating set, subgraph isomorphism, or homomorphism restricted to graphs of bounded degree are in para-LOGSPACE.

Proof of (2). The following proof requires some familiarity with finite model theory. Background on the techniques we use here can be found in [18].

We fix g. Suppose we are given a first-order formula φ and a graph $\mathcal{G} = (G, E^{\mathcal{G}})$ of degree at most g. Let n be the number of vertices of $\mathcal{G}$.

For all vertices $a, b \in G$ we let $d^{\mathcal{G}}(a, b)$ denote the distance between a and b in $\mathcal{G}$. For $r \geqslant 0$ and $a \in G$ we let

$$N_r^{\mathcal{G}}(a) := \{b \in G \mid d^{\mathcal{G}}(a, b) \leqslant r\},$$

and we let $\langle N_r^{\mathcal{G}}(a)\rangle$ be the subgraph induced by $\mathcal{G}$ on $N_r^{\mathcal{G}}(a)$.

By Gaifman's Theorem [16], we can assume that φ is of the form

$$\exists x_1 \ldots \exists x_k \Big(\bigwedge_{1 \leqslant i < j \leqslant k} d(x_i, x_j) > 2r \wedge \bigwedge_{1 \leqslant i \leqslant k} \psi(x_i) \Big),$$

where $\psi(x)$ is *r-local*, which means that for all $a \in G$ we have

$$\mathcal{G} \models \psi(a) \iff \langle N_r^{\mathcal{G}}(a)\rangle \models \psi(a). \tag{3}$$

Furthermore, $d(x_i, x_j) > 2r$ is a first-order formula saying that the distance between x_i and x_j is greater than $2r$.

Thus to check whether $\mathcal{G} \models \varphi$, we have to find out if there exist vertices $a_1, \ldots, a_k \in G$ of pairwise distance greater than $2r$ such that $\mathcal{G} \models \psi(a_i)$ for $1 \leqslant i \leqslant k$. To do this, we are only allowed to use space $c \cdot \log n$, for some constant c, and a certain amount of extra space that does not depend on n (but may depend on φ). We refer to the extra space as the *φ-space*.

We assume that the vertices are numbered in some order, so we can address each by a binary number of length $\log n$. Moreover, it makes sense to speak of "the ith vertex" in some set $S \subseteq V$.

The following claim is crucial. Note that it is not even obvious how to decide whether $d^{\mathcal{G}}(a, b) \leqslant s$ for some s depending only on φ under the memory restrictions described above.

Claim. *For every s only depending on φ and for every $a \in G$ we can compute an isomorphic copy of $\langle N_s^{\mathcal{G}}(a)\rangle$ in the φ-space* (*obeying the memory restrictions*).

The trick is to address vertices relative to a. For all $b \in G$ and $1 \leqslant i \leqslant g$, let $f_i(b)$ denote the ith neighbour of b (if b has at least i neighbours). Note that the vertex $f_i(b)$ can easily be computed in space $O(\log i + \log n)$. For every vertex $b \in N_s^{\mathcal{G}}(a)$ there exists a tuple $(i_1, \ldots, i_p) \in \{1, \ldots, g\}^p$, for some $p \leqslant s$, such that $b = f_{i_p}(\cdots f_{i_1}(a)\cdots)$. Now to copy $\langle N_s^{\mathcal{G}}(a)\rangle$ into the φ-space, we start by reserving a memory cell for every tuple $(i_1, \ldots, i_p) \in \{1, \ldots, g\}^p$ and every $p \leqslant s$. Then we check for every such tuple if there exists a corresponding vertex. For all pairs of tuples we check if the corresponding vertices are identical and, if not, if there is an edge between them. At each time, we have to store at most three vertices at the same time in our workspace. This proves the claim.

Using the claim and (3), we see that we can decide whether for a vertex a we have $\mathcal{G} \models \psi(a)$ under our memory restrictions. Let $S := \{a \in G \mid \mathcal{G} \models \psi(a)\}$. We call a set $I \subseteq S$ of vertices of pairwise distance greater than $2r$ a *scattered set*. We have to check whether there exists a scattered set of size at least k.

We first check whether there is a vertex $a \in S$ such that there is a scattered set $I \subseteq N_{4rk}^{\mathcal{G}}(a)$ of size at least k, this can easily be done using the claim. If we find such a vertex, we are done. So let us assume that it is not the case.

For every $a \in S$, let S_a be the connected component of a in the graph with vertex set S and an edges (a, b) whenever $a \neq b$ and $d^{\mathcal{G}}(a, b) \leqslant 2r$. Then $S_a \subseteq N_{4rk}^{\mathcal{G}}(a)$. To see this, suppose for contradiction that this is not the case. Then for $1 \leqslant i \leqslant 2k$ there must be at least one vertex b_i in $N_{2ri}^{\mathcal{G}}(a) \setminus N_{2r(i-1)}^{\mathcal{G}}(a)$. But then the vertices $a, b_2, b_4, \ldots, b_{2k-1}$ form a scattered set $I \subseteq N_{4rk}^{\mathcal{G}}(a)$ of size at least k, a contradiction.

By the claim, for all a, b we can actually test within our resource bounds whether $b \in S_a$, which is equivalent to $S_a = S_b$. Let k_a be the size of the largest scattered subset of S_a. Observe that there is a scattered set of size k if, and only if, there are vertices $a_1, \ldots, a_l$ such that $S_{a_i} \cap S_{a_j} = \emptyset$ for $1 \leqslant i < j \leqslant l$ and $\sum_{i=1}^{l} k_{a_i} \geqslant k$.

Now we can proceed as follows: we keep a counter (in φ-space) that tells us how many scattered vertices we have already found, initially this counter is set to 0. As soon as the counter reaches k, we stop and accept. If the counter never reaches k, we eventually reject. We process the vertices in the order of their indices. For each vertex, we check whether it belongs to S. Whenever we find an $a \in S$, we check whether S_a contains a vertex b of lower index, if this is the case, $S_a = S_b$ has already been processed, and we continue with the next a. If not, we compute k_a and add it to the counter. □

There is another natural way of associating a parameterized complexity class with a classical class. For $k \geqslant 1$, the *kth slice* of a parameterized problem $P \subseteq \Sigma^* \times \mathbb{N}$ is the problem $P_k = \{x \mid (x, k) \in P\} \subseteq \Sigma^*$.

Definition 7. Let K be a classical complexity class. Then XK is the class of all parameterized problems P all of whose slices are in K.

To compare para-K and XK and to obtain further basic facts about these classes, we only require the complexity class K to be *robust*, i.e., to satisfy the following two conditions (for all alphabets Σ and Π):

- For every problem $X \subseteq \Sigma^*$ with $X \in \mathrm{K}$ and every word $y \in \Pi^*$: $X \times \{y\} \in \mathrm{K}$.
- For every problem $X \subseteq \Sigma^* \times \Pi^*$ with $X \in \mathrm{K}$ and every word $y \in \Pi^*$: $X_y = \{x \in \Sigma^* \mid (x, y) \in X\} \in \mathrm{K}$.

Clearly, every regular class is robust. Moreover, for every robust complexity class K and every problem $X \in \mathrm{K}$, the trivial parameterization $X \times \{0\}$ of X is in para-K. And para-K $\subseteq$ XK holds for every robust complexity class K.

Proposition 8. *For robust complexity classes* K *and* K′ *the following are equivalent*:
(1) K $\subseteq$ K′.
(2) para-K $\subseteq$ para-K′.
(3) para-K $\subseteq$ XK′.
In particular, *there is no robust* K_0 *such that* para-K_0 *is between* para-K *and* XK.

Proof. The implications (1) $\Rightarrow$ (2) and (2) $\Rightarrow$ (3) are trivial. For (3) $\Rightarrow$ (1), let $Y \in K$. Then the trivial parameterization $Y \times \{0\}$ is in para-K. Thus, by the assumption para-K $\subseteq$ XK′, $Y \in \mathrm{K}'$. □

In Section 3, we shall use Proposition 8 to show that the classes of the W-hierarchy and other important parameterized complexity classes, which are between FPT = para-PTIME and XPTIME, are "inherently parametric," that is, not of the form para-K for any robust K.

Remark 9. For regular complexity classes K the containment of para-K in XK is always strict just because we defined XK in a highly non-uniform way (so that it contains problems that are not computable).

For the standard, well-behaved complexity classes K, which are all of the form $\mathrm{K}(\mathcal{M}, \mathcal{R}, \Gamma)$ for a simple class Γ, we can define a uniform version of XK (see [9]). There are classes K for which this uniform-XK coincides with para-K. An example is the class K of all linear time solvable problems. For other classes K, such as LOGSPACE, PTIME, NP, or PSPACE, a simple diagonalisation shows that para-K $\neq$ uniform-XK.

Remark 10. The previous remark highlights the issue of *uniformity*, which deserves some further discussion. Our definitions of the classes para-K and also the reductions defined in the next section are based on what Downey and Fellows [9] call *strongly uniformly fixed-parameter tractable*. Recall that according to this definition, a parameterized problem P is fixed-parameter tractable if there is a *computable* function $f : \mathbb{N} \to \mathbb{N}$, a constant c, and an algorithm deciding P in time $\mathrm{O}(f(k) \cdot n^c)$.

If we do not require f to be computable we get a notion that Downey and Fellows call *uniformly fixed-parameter tractable*. While this notion seems to be perfectly reasonable at first sight, it mixes computability and non-computability in a way that makes the class much less robust. For example, an analogon of our Theorem 4 does not seem to hold for uniform fixed-parameter tractability (actually, it is not even clear how to formulate such an analogon). We always felt that strongly uniform fixed-parameter tractability was the more natural definition and used it in our earlier papers, and we believe that the results presented here support this intuition.

There is also a non-uniform notion of fixed-parameter tractability. A problem P is *non-uniformly fixed-parameter tractable* if there is a constant c such that for every $k \in \mathbb{N}$ the kth slice P_k of P is

decidable in time $O(n^c)$. As strongly uniform fixed-parameter tractability, this notion is coherent and admits a clean theory; indeed all results of the present paper can easily be adapted to non-uniform fixed-parameter tractability (actually, this makes some of the proofs significantly simpler). There is a lot to be said in favour of the non-uniform theory, especially when it comes to classes such as XP, whose natural definition is the non-uniform one. The main backdraw of non-uniform fixed-parameter tractability is that it is not very appealing to have a complexity theory whose "tractable" problems are not even necessarily decidable.

2.1. Reductions and completeness

The following definition introduces the basic type of reduction on which much of the theory of parameterized intractability is built:

Definition 11. Let $P \subseteq \Sigma^* \times \mathbb{N}$ and $P' \subseteq (\Sigma')^* \times \mathbb{N}$ be parameterized problems.

A *para-PTIME-reduction*, or *FPT-reduction*, from P to P' is a mapping $R : \Sigma^* \times \mathbb{N} \to (\Sigma')^* \times \mathbb{N}$ such that:

(1) For all $(x, k) \in \Sigma^* \times \mathbb{N}$ we have $(x, k) \in P \iff R(x, k) \in P'$.
(2) There exists a computable function $g : \mathbb{N} \to \mathbb{N}$ such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$, say with $R(x, k) = (x', k')$, we have $k' \leqslant g(k)$.
(3) There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that R is computable in time $f(k) \cdot n^c$.

We write $P \leqslant^{\mathrm{FPT}} P'$ if there is an FPT-reduction from P to P'.

FPT-reductions are parameterized versions of polynomial time many one reductions. Of course we can also introduce parameterized analogues of other types of reductions. Instead of doing this systematically, we just introduce a parameterized analogue of LOGSPACE-reductions:

Definition 12. Let $P \subseteq \Sigma^* \times \mathbb{N}$ and $P' \subseteq (\Sigma')^* \times \mathbb{N}$ be parameterized problems.

A *para-LOGSPACE-reduction*, or *PL-reduction*, from P to P' is a mapping $R : \Sigma^* \times \mathbb{N} \to (\Sigma')^* \times \mathbb{N}$ that satisfies conditions (1) and (2) of Definition 11 and

(3′) There exists a computable function $f : \mathbb{N} \to \mathbb{N}$ and a constant $c \in \mathbb{N}$ such that R is computable in space $f(k) + c \cdot \log n$.

We write $P \leqslant^{\mathrm{PL}} P'$ if there is a PL-reduction from P to P'.

Proposition 13. *Let* K *be a complexity class that is closed under PTIME-reductions* (*LOGSPACE-reductions*, *respectively*). *Then* para-K *is closed under FPT-reductions* (*PL-reductions*, *respectively*).

Proof. We give the proof for PTIME-reductions, the statement for LOGSPACE-reductions is proved analogously.

Let $Q \subseteq \Pi^* \times \mathbb{N}$ be in para-K, $P \subseteq \Sigma^* \times \mathbb{N}$, and $P \leqslant^{\mathrm{FPT}} Q$. We have to show that P is in para-K. By the assumption $P \leqslant^{\mathrm{FPT}} Q$, there is an algorithm that, given $(x, k) \in \Sigma^* \times \mathbb{N}$, in time $\leqslant f(k) \cdot |x|^c$ computes (x', k') with

$$(x, k) \in P \iff (x', k') \in Q$$

and with $k' \leqslant g(k)$ (where f, g are suitable computable functions and $c \in \mathbb{N}$). Since $Q \in$ para-K, Q is in K after a pre-computation, i.e., there is $Y \in$ K, $Y \subseteq \Pi^* \times \Delta^*$, and a computable $p : \mathbb{N} \to \Delta^*$ such that

$$(x', k') \in Q \iff (x', p(k')) \in Y.$$

Let r be a computable function such that $r(k) \geqslant f(k)$ and such that for all $m \leqslant g(k)$, $r(k)$ is greater than the number of steps needed to compute $g(m)$.

We shall define a problem X that is PTIME-reducible to Y, $X \subseteq \Sigma^* \times (\mathbb{N} \times \mathbb{N})$, such that for arbitrary $(x, k) \in \Sigma^* \times \mathbb{N}$,

$$(x, (k, r(k)) \in X \iff (x', p(k')) \in Y.$$

Then X is in K, because K is closed under PTIME-reductions. Therefore, since

$$(x, k) \in P \iff (x', p(k')) \in Y \iff (x, (k, r(k)) \in X,$$

P is in K after a pre-computation.

X is defined via the following algorithm $\mathbb{A}$: given $(x, (k, \ell)) \in \Sigma^* \times (\mathbb{N} \times \mathbb{N})$, $\mathbb{A}$ first performs $\ell \cdot |x|^c$ steps of the computation of (x', k'); if this computation does not stop within this time, then $(x, (k, \ell)) \notin X$. Otherwise, $\mathbb{A}$ performs ℓ steps of the computation of $p(k')$. Again, if this computation does not stop, then $(x, (k, \ell)) \notin X$. Otherwise, we fix:

$$(x, (k, l)) \in X \iff (x', p(k')) \in Y.$$

Note that the algorithm $\mathbb{A}$ immediately yields a PTIME-reduction of X to Y: for this purpose choose $(y_0, z_0) \notin Y$. The PTIME-reduction assigns (y_0, z_0) to $(x, (k, l))$ if one of the "sub-computations" does not stop in the required time; otherwise, it assigns $(x', p(k'))$. □

We define *hardness* and *completeness* of a parameterized problem for a parameterized complexity class under FPT-reductions or PL-reductions in the usual way.

So what are complete problems for our classes para-K? One might guess that suitable parameterizations of complete problems for K are complete for para-K. But it is not obvious which parameterizations are suitable. It came as a surprise to us that essentially we do not need any parameterization—the trivial parameterization of a complete problem for K is complete for para-K. So the parameterized complexity classes para-K have complete problems that are essentially unparameterized.

Proposition 14. *Let* K *be a robust complexity class, and let X be complete for* K *under PTIME-reductions* (*LOGSPACE-reductions, respectively*). *Then $X \times \{0\}$ is complete for* para-K *under FPT-reductions* (*PL-reductions, respectively*).

Proof. We give the proof for LOGSPACE-reductions and PL-reductions, the statement for PTIME-reductions and FPT-reductions is proved analogously.

Let $P \subseteq \Sigma^* \times \mathbb{N}$ be in para-K. Then there is a computable function $f : \mathbb{N} \to \Pi^*$ for some alphabet Π and a problem $Y \in$ K such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ we have

$$(x, k) \in P \iff (x, f(k)) \in Y.$$

Since X is complete for K, there is a LOGSPACE-reduction from Y to X, that is, a LOGSPACE computable mapping R such that for all $(x, y) \in \Sigma^* \times \Pi^*$ we have $(x, y) \in Y \iff R(x, y) \in X$. We claim that the mapping S defined by

$$S(x, k) = (R(x, f(k)), 0)$$

is a PL-reduction from P to $X \times \{0\}$. Clearly, we have

$$(x, k) \in P \iff S(x, k) \in X \times \{0\}.$$

It is also not hard to see that, given (x, k), $S(x, k)$ can be computed in space $g(k) + O(\log n)$ for a suitable function g. The other conditions are trivially satisfied. □

The following proposition shows that completeness for a class para-K is always based on a somewhat trivial parameterization.

Proposition 15. *Let* K *be a robust complexity class, and let* $P \subseteq \Sigma^* \times \mathbb{N}$ *be complete for* para-K *under FPT-reductions* (*PL-reductions*, *respectively*).

Then there is an integer $\ell \in \mathbb{N}$ *such that the problem* $P \cap (\Sigma^* \times \{0, \ldots, \ell\})$ *is complete for* para-K *under FPT-reductions* (*PL-reductions*, *respectively*).

Proof. We give the proof for PL-reductions, the statement for FPT-reductions is proved analogously.

Let $P \subseteq \Sigma^* \times \mathbb{N}$ be complete for para-K under PL-reductions. Then $P \in$ para-K, so there exists a computable function $f : \Sigma^* \to \Pi^*$ for some alphabet Π and a problem $Y \in$ K such that for all $(x, k) \in \Sigma^* \times \mathbb{N}$ we have

$$(x, k) \in P \iff (x, f(k)) \in Y.$$

Since P is hard for para-K, for all $Q \in$ para-K we have $Q \leqslant^{\mathrm{PL}} P$. Thus in particular, for all $X \in K$ we have

$$X \times \{0\} \leqslant^{\mathrm{PL}} P. \tag{4}$$

As a first step, we prove that Y is complete for K under LOGSPACE-reductions. Let $X \subseteq \Delta^*$ be in K. (4) implies that there exists an $\ell \in \mathbb{N}$ ($\ell = g(0)$ in the terminology of Definition 12) and LOGSPACE-computable mappings $R_1 : \Delta^* \to \Sigma^*$, $R_2 : \Delta^* \to \{0, \ldots, \ell\}$ such that for all $x \in \Delta^*$ we have

$$x \in X \iff (R_1(x), R_2(x)) \in P.$$

Thus

$$x \in X \iff (R_1(x), f(R_2(x))) \in Y.$$

Since the finite mapping $f|_{\{0,\ldots,\ell\}}$ is LOGSPACE-computable, the mapping

$$x \mapsto (R_1(x), f(R_2(x)))$$

is a LOGSPACE-reduction from X to Y. This shows that Y is complete for K under LOGSPACE-reductions.

By (4), we have $Y \times \{0\} \leqslant^{\mathrm{PL}} P$. A similar argument as above shows that there exists an $\ell \in \mathbb{N}$ such that actually $Y \times \{0\} \leqslant^{\mathrm{PL}} P_0 := P \cap (\Sigma^* \times \{0, \ldots, \ell\})$. Because, by Proposition 14, $Y \times \{0\}$ is complete for para-K under PL-reductions, we have $Q \leqslant^{\mathrm{PL}} P_0$ for all $Q \in$ para-K. Thus P_0 is complete for para-K under PL-reductions. □

Remark 16. We actually get a bit more out of the proof of Proposition 15. First of all, the proof shows that if para-K has a complete problem P then K also has a complete problem. Moreover, it shows that

$P_0 := P \cap (\Sigma^* \times \{0, \ldots, \ell\})$, considered as a classical problem, is hard for K under PTIME-reductions (LOGSPACE-reductions, respectively). Thus if $P_0 \in$ K, which is, for example, the case if K is closed under finite unions, then P_0 is complete for K under the respective reductions.

At first sight, Propositions 14 and 15 seem to indicate that the classes para-K are trivial from the perspective of parameterized complexity theory. Of course this is not the case, these classes contain problems with interesting parameterizations. After all, the most important parameterized complexity class FPT is of this form. As another example, we have seen that p-MC(DEG(g), FO) $\in$ para-LOGSPACE, for every $g \geqslant 1$. It is worthwhile to note that the trivial parameterization of MC(DEG(g), FO) is complete for para-PSPACE under PL-reductions (because MC(DEG(g), FO) is complete for PSPACE under LOGSPACE-reductions [27]). Since LOGSPACE $\subset$ PSPACE and thus para-LOGSPACE $\subset$ para-PSPACE, this implies that there is no PL-reduction from MC(DEG(g), FO) $\times \{0\}$ to p-MC(DEG(g), FO).

2.2. Logical descriptions

In this section, we show how to derive logical descriptions of the classes para-K from such descriptions of the corresponding K.

2.2.1. Preliminaries from logic and descriptive complexity theory

A *vocabulary* is a finite set of relation, function, and constant symbols. Each relation and function symbol has an *arity*. τ always denotes a vocabulary. A *structure* $\mathcal{A}$ of vocabulary τ, or τ-structure, consists of a set A called the universe, and an interpretation $T^{\mathcal{A}}$ of each symbol $T \in \tau$: relation symbols and function symbols are interpreted by relations and functions on A of the appropriate arity, and constant symbols are interpreted by elements of A. *We only consider structures whose universe is finite.* An *ordered structure* is a structure $\mathcal{A}$ whose vocabulary contains the binary relation symbol $\leqslant$, the unary function symbol S, and constant symbols min and max, such that $\leqslant^{\mathcal{A}}$ is a linear order of A, $\min^{\mathcal{A}}$ and $\max^{\mathcal{A}}$ are the minimum and maximum element of $\leqslant^{\mathcal{A}}$, and $S^{\mathcal{A}}$ is the successor function associated with $\leqslant^{\mathcal{A}}$, where we let $S^{\mathcal{A}}(\max^{\mathcal{A}}) = \max^{\mathcal{A}}$. To avoid notational overkill, we often omit superscripts $^{\mathcal{A}}$.

By $S^k(\min)$ we denote the term

$$\underbrace{S(S(\ldots S}_{k \text{ times}}(\min)\ldots)),$$

which is interpreted by the kth element of the ordering. By ORD we denote the class of all ordered structures, and for a vocabulary τ, by ORD[τ] we denote the class of all ordered $\tau^{\text{ord}} := \tau \cup \{\leqslant, S, \min, \max\}$-structures. *In the following, we assume that vocabularies do not contain any function symbols besides the successor function S.*

In descriptive complexity theory, (classical) problems are identified with classes of ordered structures. A canonical way to do this is by associating with each word $w \in \Sigma^*$ an ordered structure $\mathcal{W}$ whose vocabulary contains a unary relation symbol P_a for each letter $a \in \Sigma$. Then a problem $X \subseteq \Sigma^*$ corresponds to a subclass of ORD[$\{P_a \mid a \in \Sigma\}$]. It is one of the nice aspects of this framework that usually we can encode problems in a more direct way than going through words. For example, we can identify graph problems with classes of (ordered) $\{E\}$-structures, where E is a binary relation symbol. Until the end of this section, and also in Subsection 3.1, classical problems are subclasses of ORD[τ], for a vocabulary τ, that are closed under isomorphism. Similarly, parameterized problems are subclasses P

of $\mathrm{ORD}[\tau] \times \mathbb{N}$ for a vocabulary τ, where for each $k \in \mathbb{N}$ the class $P_k = \{\mathcal{A} \in \mathrm{ORD}[\tau] \mid (\mathcal{A}, k) \in P\}$ is closed under isomorphism.

We assume that the reader is familiar with first-order logic. The class of all formulas of first-order logic is denoted by FO.

If $\varphi(x_1, \ldots, x_k)$ is a formula with free variables among $x_1, \ldots, x_k$, $\mathcal{A}$ is a structure, and $a_1, \ldots, a_k \in A$, then we write $\mathcal{A} \models \varphi(a_1, \ldots, a_k)$ to denote that $\mathcal{A}$ satisfies φ if the variables $x_1, \ldots, x_k$ are interpreted by $a_1, \ldots, a_k$, respectively. We let $\varphi(\mathcal{A}) = \{(a_1, \ldots, a_k) \in A^k \mid \mathcal{A} \models \varphi(a_1, \ldots, a_k)\}$. Recall that a *sentence* is a formula without free variables. Sentences define classes of structures. In particular, a sentence φ whose vocabulary is contained in τ^{ord} for some vocabulary τ defines the class

$$X(\varphi) = \{\mathcal{A} \in \mathrm{ORD}[\tau] \mid \mathcal{A} \models \varphi\} \subseteq \mathrm{ORD}[\tau]$$

of ordered structures. We say that a logic L *captures* a complexity class K, and write $\mathrm{K} = \mathrm{L}$, if for every vocabulary τ and every problem $X \subseteq \mathrm{ORD}[\tau]$ we have:

$$X \in \mathrm{K} \iff \text{There exists a sentence } \varphi \in \mathrm{L}[\tau^{\mathrm{ord}}] \text{ such that } X = X(\varphi).$$

Here a *logic* is simply a class of formulas (together with a semantics for these formulas). For a logic L we denote by $\mathrm{L}[\tau]$ the class of formulas of L whose vocabulary is contained in τ.

The following theorem summarises the most important results from descriptive complexity theory:

Theorem 17 (cf. [14,21,22,28])**.**
(1) *Deterministic transitive closure logic* DTC *captures* LOGSPACE.
(2) *Transitive closure logic* TC *captures* NLOGSPACE.
(3) *Least fixed-point logic* LFP *captures* PTIME.
(4) *Existential second-order logic* Σ^1_1 *captures* NP. *Moreover, for every* $t \geqslant 2$, Σ^1_t *captures* Σ^{P}_t.
(5) *Partial fixed-point logic* PFP *captures* PSPACE.

It would not make much sense to give a long list of definitions for all these logics here, and there really is no need to do so, because in our proofs we basically just apply Theorem 17 (and a number of related technical lemmas) together with the results of the previous section. Let us just give a brief description of the logics here: transitive closure logic is an extension of first-order logic by an operator that essentially formalises the directed graph reachability problem, and deterministic transitive closure logic is a variant using deterministic reachability, that is, reachability through nodes of out-degree 1. Least fixed-point logic extends first-order logic by a monotone fixed-point operator, and partial fixed-point logic extends first-order logic by an arbitrary fixed-point operator. Second-order logic allows not only quantification over elements of the universe of a structure, but also over relations on the universe. Σ^1_t is the fragment of second-order logic whose formulas have at most t alternating blocks of existential and universal second-order quantifiers, starting with a block of existential quantifiers (see [12] or [23] for details).

2.2.2. Capturing parameterized complexity classes

Recall that we view parameterized problems as subclasses of $\mathrm{ORD}[\tau] \times \mathbb{N}$.

Definition 18. Let L be a logic.
(1) A parameterized problem $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ is *slicewise* L-*definable*, if there is a computable function $\delta : \mathbb{N} \to \mathrm{L}[\tau^{\mathrm{ord}}]$ such that for all $\mathcal{A} \in \mathrm{ORD}[\tau]$ and $k \in \mathbb{N}$ we have

$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \delta(k)$.

(2) L *captures* a parameterized complexity class Q, if for every vocabulary τ and every parameterized problem $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ we have

$P \in \mathrm{Q} \iff P$ is slicewise L-definable.

If this is the case, we write Q = slicewise-L.

For well-behaved complexity classes K and logics L with K = L this equality holds "effectively"; therefore, in view of Remark 9,

$\mathrm{K} = \mathrm{L} \iff$ uniform-XK = slicewise-L.

Proposition 19. *Let* K *be a classical complexity class and* Q *a parameterized complexity class. If* para-K $\subseteq$ Q $\subset$ uniform-XK *then there is no logic* L *with* Q = *slicewise*-L.

Proof. We assume Q = slicewise-L and show K = L (and hence, uniform-XK = slicewise-L, a contradiction). For $\varphi \in \mathrm{L}[\tau^{\mathrm{ord}}]$ define the parameterized problem $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ by

$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \varphi$.

Then, P is slicewise L-definable, thus $P \in$ Q. Therefore, $P_k \in$ K, i.e., $X(\varphi) \in$ K. Conversely, assume $X \subseteq \mathrm{ORD}[\tau]$ and $X \in$ K. Then $X \times \{0\} \in$ para-K $(\subseteq$ Q$)$. As Q = slicewise-L, there is a $\varphi \in \mathrm{L}[\tau^{\mathrm{ord}}]$ with $X(\varphi) = X$. □

Remark 9 shows that most of the interesting parameterized complexity classes (including FPT) are not of the form uniform-XK. Thus, by Proposition 19, if we want to describe any of these by logics, we need a more refined notion of capturing.

Definition 20. A family $(\mathrm{L}_s)_{s\in\mathbb{N}}$ of logics *captures* a parameterized complexity class Q if for every vocabulary τ and every parameterized problem $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ we have

$P \in \mathrm{Q} \iff$ There is an $s \geqslant 1$ such that P is slicewise L_s-definable.

If this is the case, we write $\mathrm{Q} = \bigcup_{s\geqslant 1}$ slicewise-L_s.

The following result shows, for example for K = PSPACE and Q = para-PSPACE, that

$$\text{para-PSPACE} = \bigcup_{s\geqslant 1} \text{slicewise-L}_s$$

implies $\bigcup_{s\geqslant 1} \mathrm{L}_s \equiv$ PFP, i.e., $\bigcup_{s\geqslant 1} \mathrm{L}_s$ and PFP have the same expressive power on ordered structures.

Proposition 21. *Let* K *be a classical complexity class and* Q *a parameterized complexity class with*
(1) Q $\subseteq$ XK.
(2) *If* $X \in$ K, *then* $X \times \{0\} \in$ Q.
Moreover, let L_s *for* $s \geqslant 1$ *be logics with* $\mathrm{Q} = \bigcup_{s\geqslant 1}$ *slicewise*-L_s. *Then* $\bigcup_{s\geqslant 1} \mathrm{L}_s$ *captures* K.

Proof. First, assume that $\varphi \in \bigcup_{s\geqslant 1} \mathrm{L}_s$. Then, $X(\varphi) \times \mathbb{N} \in$ Q and hence by (1), $X(\varphi) \in$ K. Conversely, let $Y \subseteq \mathrm{ORD}[\tau]$ be a class in K. Then by (2), $Y \times \{0\} \in$ Q, so by the assumption $\mathrm{Q} = \bigcup_{s\geqslant 1}$

slicewise-L_s, there is an $s \geqslant 1$ and $\delta : \mathbb{N} \to \mathrm{L}_s[\tau^{\mathrm{ord}}]$ with $((\mathcal{A}, k) \in Y \times \{0\} \iff \mathcal{A} \models \delta(k))$; thus, $Y = X(\delta(0))$. $\square$

So to capture the parameterized versions of the complexity classes mentioned in Theorem 17 we have to look for suitable "filtrations" of the corresponding logics. Recall that the quantifier rank of a first-order formula is the maximum nesting depth of quantifiers in the formula. For $s \in \mathbb{N}$, we denote by FO_s the set of first-order formulas of quantifier rank at most s and by FO^s the set of first-order formulas that contain at most s variables.

It is well known that every formula of least fixed-point logic LFP is equivalent to one of the form

$$[\mathrm{LFP}_{\bar{x},X}\, \varphi]\, \bar{t}, \tag{5}$$

where φ is a first-order formula and $\bar{t}$ a tuple of terms [21]. LFP^s and LFP_s denote the fragments of LFP consisting of all formulas of the form (5) where $\varphi \in \mathrm{FO}^s$ and $\varphi \in \mathrm{FO}_s$, respectively, and where $\bar{t}$ has length at most s.

On ordered structures, every formula of transitive closure logic TC is equivalent to one of the form

$$[\mathrm{TC}_{\bar{u},\bar{v}}\, \varphi]\, \bar{t}_1, \bar{t}_2 \tag{6}$$

where φ is a first-order formula and $\bar{t}_1, \bar{t}_2$ are tuples of terms [22]. TC_s denotes the fragment of TC consisting of all formulas of the form (6) where $\varphi \in \mathrm{FO}_s$ and $\bar{t}_1, \bar{t}_2$ are tuples of length at most s.

$(\Sigma^1_1)^s$ and $(\Sigma^1_1)_s$ are the sets of formulas of the form $\exists X_1 \ldots \exists X_m\, \varphi$ with $\varphi \in \mathrm{FO}^s$ or $\varphi \in \mathrm{FO}_s$, respectively. Similarly, we can define fragments DTC_s, $(\Sigma^1_t)^s$ and $(\Sigma^1_t)_s$, PFP^s, and PFP_s of the respective logics.

We have proved a first parameterized capturing result in [15]. The second equality in the following theorem has not been stated explicitly in [15], but follows easily from the proof.

Theorem 22 (cf. [15])**.** $\mathrm{FPT} = \bigcup_{s\geqslant 1} \text{slicewise-}\mathrm{LFP}^s = \bigcup_{s\geqslant 1} \text{slicewise-}\mathrm{LFP}_s$.

The following theorem lifts the other statements of Theorem 17 to the parameterized classes:

Theorem 23.
(1) $\text{para-LOGSPACE} = \bigcup_{s\geqslant 1} \text{slicewise-DTC}_s$.
(2) $\text{para-NLOGSPACE} = \bigcup_{s\geqslant 1} \text{slicewise-TC}_s$.
(3) $\text{para-NP} = \bigcup_{s\geqslant 1} \text{slicewise-}(\Sigma^1_1)^s = \bigcup_{s\geqslant 1} \text{slicewise-}(\Sigma^1_1)_s$,
and for every $t \geqslant 2$, $\text{para-}\Sigma^{\mathrm{P}}_t = \bigcup_{s\geqslant 1} \text{slicewise-}(\Sigma^1_t)^s = \bigcup_{s\geqslant 1} \text{slicewise-}(\Sigma^1_t)_s$.
(4) $\text{para-PSPACE} = \bigcup_{s\geqslant 1} \text{slicewise-}\mathrm{PFP}^s = \bigcup_{s\geqslant 1} \text{slicewise-}\mathrm{PFP}_s$.

Before we turn to a proof of this theorem, we mention the following facts that are well known and easily proved. To prove Lemma 25, just recall that we have a successor function S.

Lemma 24. *Let τ be a vocabulary and $s \geqslant 1$.*
(1) *There is an $O(|\varphi| \cdot n^s)$-algorithm that decides for every sentence $\varphi \in \mathrm{FO}^s[\tau^{\mathrm{ord}}]$ and every $\mathcal{A} \in \mathrm{ORD}[\tau]$ if $\mathcal{A}$ satisfies φ.*
(2) *There is a space $O(|\varphi| + \log n)$-algorithm that decides for every sentence $\varphi \in \mathrm{FO}_s[\tau^{\mathrm{ord}}]$ and every $\mathcal{A} \in \mathrm{ORD}[\tau]$ if $\mathcal{A}$ satisfies φ.*

Lemma 25. *Let τ be a vocabulary. For every structure $\mathcal{A} \in \mathrm{ORD}[\tau]$ there is a sentence $\varphi_{\mathcal{A}} \in \mathrm{FO}^0 \cap \mathrm{FO}_0$, that is, a sentence with no variables and therefore no quantifiers, such that for all $\mathcal{B} \in \mathrm{ORD}[\tau]$ we have:*

$$\mathcal{B} \models \varphi_{\mathcal{A}} \Longleftrightarrow \mathcal{A} \cong \mathcal{B}.$$

Proof (of Theorem 23). By standard means one shows that if the parameterized problem $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ is slicewise definable in one of the logics mentioned on the right-hand sides of the equalities, then P is in the corresponding complexity class. We exemplify this for para-NLOGSPACE. So, assume that P is slicewise TC_s-definable (for some fixed s) via $\delta : \mathbb{N} \to \mathrm{TC}_s[\tau^{\mathrm{ord}}]$. Given $\mathcal{A} \in \mathrm{ORD}[\tau]$ and $k \in \mathbb{N}$, one computes $\delta(k) \in \mathrm{TC}_s$, say

$$\delta(k) = [\mathrm{TC}_{\bar{u},\bar{v}}\, \varphi]\, \bar{t}_1, \bar{t}_2,$$

where the quantifier rank of φ is $\leqslant s$ and $|\bar{u}| = m \leqslant s$. Then

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models [\mathrm{TC}_{\bar{u},\bar{v}}\, \varphi]\, \bar{t}_1, \bar{t}_2,$$

i.e., $(\mathcal{A}, k) \in P$ if and only if there is a sequence $\bar{a}_1, \bar{a}_2, \ldots, \bar{a}_m$ of tuples of elements of A such that $\bar{a}_1$ is the interpretation of $\bar{t}_1$ in $\mathcal{A}$, $\bar{a}_m$ is the interpretation of $\bar{t}_2$ in $\mathcal{A}$, and for $1 \leqslant i \leqslant m-1$ we have $\mathcal{A} \models \varphi(\bar{a}_i, \bar{a}_{i+1})$. We call such a sequence a φ-path from $\bar{t}_1$ to $\bar{t}_2$. Using part (2) of Lemma 24, it is easy to see that there is a a non-deterministic algorithm $\mathbb{A}_k$ that, given a structure $\mathcal{A}$ of size n, decides whether there is a φ-path from $\bar{t}_1$ to $\bar{t}_2$ in space $O(|\varphi| + \log n)$. Thus P is decidable in non-deterministic space $g(k) + c \cdot \log n$, where c is a suitable constant and $g(k)$ is $c \cdot |\delta(k)|$ plus the space needed to compute $\delta(k)$ and to obtain $\mathbb{A}_k$.

For the converse inclusions, let K be one of the classes appearing on the left-hand side of the equalities, and let $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ be a parameterized problem in para-K. By Theorem 4, P is eventually in K, that is, there is a computable function $f : \mathbb{N} \to \mathbb{N}$ and a problem $X \in \mathrm{K}$ such that for all $(\mathcal{A}, k) \in \mathrm{ORD}[\tau] \times \mathbb{N}$ with $n > f(k)$ we have

$$(\mathcal{A}, k) \in P \Longleftrightarrow (\mathcal{A}, k) \in X.$$

Without loss of generality we assume $f(k) \geqslant k$.

Let c be a new constant symbol. For $\mathcal{A} \in \mathrm{ORD}[\tau]$ and $k \in \mathbb{N}$ with $k < |\mathcal{A}|$, we denote by $\langle \mathcal{A}, k \rangle$ the $\tau^{\mathrm{ord}} \cup \{c\}$-structure obtained from $\mathcal{A}$ by interpreting c by the kth element of the order in $\mathcal{A}$; in particular, $\langle \mathcal{A}, 0 \rangle \models c = \min$. Let

$$Y := \{\langle \mathcal{A}, k \rangle \mid (\mathcal{A}, k) \in X \text{ and } k < |\mathcal{A}|\}.$$

Clearly, $Y \in \mathrm{K}$. Therefore, there is a $\tau^{\mathrm{ord}} \cup \{c\}$-sentence φ of the logic capturing K such that

$$\langle \mathcal{A}, k \rangle \in Y \iff \langle \mathcal{A}, k \rangle \models \varphi.$$

Thus, for $(\mathcal{A}, k) \in \mathrm{ORD}[\tau] \times \mathbb{N}$ with $n > f(k)$ we have

$$(\mathcal{A}, k) \in P \iff \langle \mathcal{A}, k \rangle \models \varphi.$$

At this point we have to consider each complexity class separately, although the arguments more or less remain the same. For example, let us assume that $\mathrm{K} = \mathrm{NP}$. Then the sentence φ has the form

$$\varphi = \exists X_1 \ldots \exists X_m\, \psi,$$

where ψ contains no second-order quantifiers and, say, contains s first-order quantifiers. Thus $\psi \in \mathrm{FO}^s \cap \mathrm{FO}_s$. Then, for all $(\mathcal{A}, k) \in \mathrm{ORD}[\tau] \times \mathbb{N}$ we have

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \delta_k,$$

where

$$\delta_k := \exists X_1 \ldots \exists X_m \Big(\big(S^{f(k)-1}(\min) \neq \max \wedge \varphi(S^k(\min)/c) \big) \vee \bigvee_{(\mathcal{A},k)\in P,\, |\mathcal{A}| \leqslant f(k)} \varphi_{\mathcal{A}} \Big).$$

Here, $\varphi(S^k(\min)/c))$ is obtained from φ by substituting c by $S^k(\min)$ and the $\varphi_{\mathcal{A}}$ are formulas in $\mathrm{FO}^0 \cap \mathrm{FO}_0$ chosen according to Lemma 25. Note that $S^{f(k)-1}(\min) \neq \max$ implies that the size of the structure is greater than $f(k)$. Thus $\delta : \mathbb{N} \to (\Sigma_1^1)^s[\tau^{\mathrm{ord}}]$ with $\delta(k) := \delta_k$ is both a slicewise $(\Sigma_1^1)^s$-definition of P and a slicewise $(\Sigma_1^1)_s$-definition of P.

The other cases are treated similarly. □

Remark 26. The reason that we do not get finite-variable versions of the results for LOGSPACE and NLOGSPACE (i.e., versions using appropriately defined (D)TC^s) is that we do not know whether parameterized model-checking for the finite-variable fragments FO^s of first-order logic is in para-NLOGSPACE.

3. The W-hierarchy and other "inherently parametric" classes

In this section, we shall study the classes W[t], for $t \geqslant 1$, forming the so-called *W-hierarchy*, the classes A[t], for $t \geqslant 1$, forming the *A-hierarchy*, and the class AW[$*$]. Each of these classes is defined as the closure of a particular problem or family of problems under FPT-reductions. For W[t], the defining family of problems consists of parameterized versions of the satisfiability problems for circuits of *weft* t (and varying depth). For A[t], the defining problem is a parameterized version of the halting problem for alternating Turing machines with t alternations. For AW[$*$], the defining problem is a parameterized version of the satisfiability problem for quantified Boolean formulas.

Instead of giving these definitions in more detail, we characterise all the classes in a uniform way by complete model-checking problems for fragments of first-order logic. We need some notation: recall that FO denotes the class of all first-order formulas. For $t \geqslant 1$, we let Σ_t be the class of all first-order formulas of the form

$$\exists x_{11} \ldots \exists x_{1\ell_1}\, \forall x_{21} \ldots \forall x_{2\ell_2}\, \exists x_{31} \ldots \exists x_{3\ell_3}\, \ldots\, Q x_{t1} \ldots Q x_{t\ell_t}\, \theta, \tag{7}$$

where θ is a quantifier-free formula and $Q = \exists$ if t is odd, $Q = \forall$ if t is even. For $u \geqslant 1$, we let $\Sigma_{t,u}$ be the class of all Σ_t formulas of the form (7) where $\ell_2, \ell_3, \ldots, \ell_t \leqslant u$. Note that $\Sigma_1 = \Sigma_{1,1}$, but $\Sigma_t \neq \Sigma_{t,u}$ for all $t \geqslant 2$, $u \geqslant 1$.

Recall from Example 6 that for a class D of structures and a class Φ of formulas, p-MC(D, Φ) denotes the model-checking problem for structures from D and sentences from Φ parameterized by the input formula, that is, the problem

$$\big\{ (\mathcal{A}, [\varphi]) \;\big|\; \mathcal{A} \in D, \varphi \in \Phi, \mathcal{A} \models \varphi \big\},$$

where $[\varphi]$ is a natural number encoding of φ. If D is the class of all structures, we denote p-MC(D, Φ) by p-MC(Φ).

Theorem 27 [10,11,15]**.**
Let τ be a vocabulary that contains at least one binary relation symbol.
(1) *For all $t \geqslant 1$ and every parameterized problem P we have*

$$P \in \mathrm{W}[t] \iff \textit{There exists a } u \geqslant 1 \textit{ such that } P \leqslant^{\mathrm{FPT}} \text{p-MC}(\Sigma_{t,u}[\tau]).$$

(2) *For all $t \geqslant 1$,* p-MC$(\Sigma_t[\tau])$ *is complete for* A$[t]$ *under FPT-reductions.*
(3) p-MC(FO$[\tau]$) *is complete for* AW$[*]$ *under FPT-reductions.*

Since for every fixed sentence $\varphi \in$ FO there is a polynomial time algorithm deciding whether a given structure satisfies φ, we have p-MC(FO) $\in$ XPTIME. Thus all of the classes W$[t]$, A$[t]$, and AW$[*]$ are contained in XPTIME. On the other hand, since they are closed under FPT-reductions, all of these classes contain FPT. Thus by Proposition 8, none of these classes Q is of the form para-K for a closed classical complexity class K, unless Q = FPT.

It is easy to see that for all $t, u \geqslant 1$ the problem MC$(\Sigma_{t,u})$ is in NP. Thus p-MC$(\Sigma_{t,u}) \in$ para-NP and therefore by Proposition13, W$[t] \subseteq$ para-NP. Unless PTIME = NP we actually have W$[t] \subset$ para-NP. To this, suppose that W$[t]$ = para-NP. Then W$[t]$ = FPT by the considerations of the previous paragraph. But this implies FPT = para-PTIME = para-NP and thus, by Proposition 8, PTIME = NP.

Similarly, the class A$[t]$ is easily seen to be contained in Σ_t^{P}, the tth level of the polynomial hierarchy, and unless PTIME = NP, this containment is strict. We do not know if A$[t] \subseteq \Sigma_{t-1}^{\mathrm{P}}$ for any $t \geqslant 2$, but we conjecture that this is not the case. Finally, we have AW$[*] \subseteq$ para-PSPACE, and unless PTIME = PSPACE this containment is strict.

Since $\Sigma_1 = \Sigma_{1,u}$ for every $u \geqslant 1$, we have A[1] = W[1]. It is not known whether A$[t]$ = W$[t]$ for any $t \geqslant 2$. The following example may help to understand the difference between A[2] and W[2] on an intuitive level. Downey and Fellows [8] proved that the natural parameterization of the dominating set problem is complete for W[2]. We give a similar graph theoretic problem that is complete for A[2].

We assume that a computable bijective pairing function $\langle\ ,\ \rangle : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ is given. Then, the *parameterized dominating clique problem* DC is defined by

$$(\mathcal{G}, \langle k, l\rangle) \in \mathrm{DC} \iff \begin{array}{l}\mathcal{G} \text{ is a graph that contains a set of at most } k \text{ vertices}\\ \text{which dominates every clique of size } l.\end{array}$$

Let X and Y be sets of vertices of the graph $\mathcal{G} = (G, E^{\mathcal{G}})$. Y is a *clique*, if $E^{\mathcal{G}}ab$ holds for all $a, b \in Y$, $a \neq b$. *X dominates Y*, if there are $a \in X$ and $b \in Y$ such that $E^{\mathcal{G}}ab$. A vertex *a dominates Y*, if $\{a\}$ dominates Y.

Theorem 28. DC *is complete for* A[2] *under* FPT*-reductions.*

In comparison, for every fixed $u \geqslant 1$ the problem of deciding whether a graph contains a set of k vertices that dominates every clique of size u, parameterized by k, is in W[2].

Proof. Since the $\langle k, l\rangle$-slice of DC is definable by the Σ_2-sentence

$$\exists x_1 \ldots \exists x_k \forall y_1 \ldots \forall y_l \Big(\Big(\bigwedge_{1 \leqslant i < j \leqslant l} E y_i y_j \Big) \to \bigvee_{i=1}^{k} \bigvee_{j=1}^{l} E x_i y_j \Big),$$

by Theorem 27 we have DC $\in$ A[2].

We turn to the A[2]-hardness of DC. It has been proved in [15] that p-MC(GRAPH, Σ_2) is complete for A[2] under FPT-reductions. Here GRAPH denotes the class of all undirected, loop-free graphs, considered as $\{E\}$-structures. Therefore, to obtain the A[2]-hardness of DC it would suffice to show that p-MC(GRAPH, Σ_2) $\leqslant^{\mathrm{FPT}}$ DC. It is easier to understand and to present the main idea of such a reduction if we show that

$$\text{p-MC(GRAPH, } \Sigma_2) \leqslant^{\mathrm{FPT}} \mathrm{DC}^{\mathrm{dir}},$$

where $\mathrm{DC}^{\mathrm{dir}}$ is the parameterized dominating clique problem for *directed* graphs. Thereby, we take over the definition of "Y is a clique" and of "X dominates Y" from the undirected case. At the end of the proof we mention the changes that are necessary for undirected graphs.

So, let a graph $\mathcal{G}$ and a Σ_2-sentence φ be given. We may assume (by a pre-computation) that

$$\varphi = \exists x_1 \ldots \exists x_k \bigwedge_{1 \leqslant i \leqslant r} \forall y_1 \ldots \forall y_l \bigvee_{1 \leqslant j \leqslant s} \psi_{ij}$$

with $l \geqslant 2$ and with literals ψ_{ij}, each of them containing exactly two variables. Set $\Psi_i := \{\psi_{ij} \mid j = 1, \ldots, s\}$ for $i = 1, \ldots, r$. We construct a directed graph $\mathcal{G}'$—within the time required by the definition of FPT-reduction—such that

$$\mathcal{G} \models \varphi \quad \text{iff} \quad \left(\mathcal{G}', \left\langle k + \binom{k}{2}, l \right\rangle\right) \in \mathrm{DC}. \tag{8}$$

Given a directed graph $\mathcal{H} = (H, E^{\mathcal{H}})$ and a set Z of vertices of $\mathcal{H}$, by *we add an l-clique to enforce that an element of Z is chosen* we mean: for new elements $c_1, \ldots, c_l$ we pass to a directed graph

$$\mathcal{H}' := \Big(H \cup \{c_1, \ldots, c_l\}, E^{\mathcal{H}} \cup \{(c_i, c_j) \mid i, j = 1, \ldots, l,\ i \neq j\} \cup \{(z, c_i) \mid z \in Z,\ i = 1, \ldots, l\} \Big).$$

(Clearly, then every set of vertices of $\mathcal{H}'$ dominating all l-cliques must contain an element of Z.)

The set of vertices of the graph $\mathcal{G}'$, we aim at, consists of two parts: the $\exists$-part and the $\forall$-part that take care of the existential and of the universal quantifiers in φ, respectively. The $\exists$-part contains the vertices in

$$\bigcup_{u=1}^{k} \{u\} \times G \ \cup \bigcup_{1 \leqslant u < v \leqslant k} \{(u, v)\} \times G^2.$$

For $u = 1, \ldots, k$ we add an l-clique to enforce that an element of $\{u\} \times G$ is chosen; similarly, for $1 \leqslant u < v \leqslant k$ we add an l-clique to enforce that an element of $\{(u, v)\} \times G^2$ is chosen. Moreover, we add l-cliques to enforce that

the element (u, v, a, b) is chosen, if (u, a) and (v, b) have been chosen.

This can be achieved by adding, for $u \neq v$ and $a, b \in G$, an l-clique $C_{\{(u,a),(v,b)\}}$ and by adding the edges

$$\begin{array}{ll} ((u,a),c) & \text{for } u=1,\dots,k,\ a\in G, \text{ and } c\in C_{\{(u,a'),(v,b)\}} \text{ for} \\ & \text{some } v\neq u \text{ and some } a',b\in G \text{ with } a'\neq a, \\ ((u,v,a,b),c) & \text{for } u,v=1,\dots,k,\ u<v,\ a,b\in G \text{ and } c\in \\ & C_{\{(u,a),(v,b)\}}. \end{array} \tag{9}$$

Hence, already now, we know that every subset of G' of size $k+\binom{k}{2}$ dominating all l-cliques must have the form

$$D(a_1,\dots,a_k) := \{(u,a_u) \mid u=1,\dots,k\} \cup \{(u,v,a_u,a_v) \mid 1\leqslant u<v\leqslant k\}. \tag{10}$$

We turn to the $\forall$-part. Its set of vertices is

$$\bigcup_{i=1}^{r}\bigcup_{t=1}^{l} \{i\}\times G\times\{t\}.$$

Here (i,a,t) "represents" the selection of the ith conjunct of φ and of a as the interpretation of y_t. We come to the edges:

for $i=1,\dots,r$ and $t,t'=1,\dots,l$ with $t\neq t'$, and $a,b\in G$

$$\text{we add the edge } ((i,a,t),(i,b,t')) \quad \text{iff} \quad \mathcal{G}\models \bigwedge_{\psi\in\Psi_i,\ \psi=\psi(y_t,y_{t'})} \psi(a,b).$$

Thus, the $\forall$-part is an (undirected) graph that contains a clique of size l if and only if there are $i\in\{1,\dots,r\}$ and $a_1,\dots,a_l\in G$ satisfying no literal in Ψ_i with variables among $y_1,\dots,y_l$.

Finally, the following edges between the $\exists$-part and the $\forall$-part are added:

- for $u=1,\dots,k$, $i=1,\dots,r$, $t=1,\dots,l$, and $a,c\in G$

$$\text{the edge } ((u,a),(i,c,t)) \quad \text{iff} \quad \mathcal{G}\models \bigvee_{\psi\in\Psi_i,\ \psi=\psi(x_u,y_t)} \psi(a,c);$$

- for $u,v=1,\dots,k$ with $u<v$, for $i=1,\dots,r$, $t=1,\dots,l$, and $a,b,c\in G$

$$\text{the edge } ((u,v,a,b),(i,c,t)) \text{ iff } \mathcal{G}\models \bigvee_{\psi\in\Psi_i,\ \psi=\psi(x_u,x_v)} \psi(a,b). \tag{11}$$

Now one verifies (8): by (10), every set of cardinality $k+\binom{k}{2}$ dominating all l-cliques of $\mathcal{G}'$ must have the form $D(a_1,\dots,a_k)$ for suitable $a_1,\dots,a_k$. And, by the last part of the construction, we see that

$$\mathcal{G}\models \bigwedge_{1\leqslant i\leqslant r} \forall y_1\dots\forall y_l \bigvee_{1\leqslant j\leqslant s} \psi_{ij}(a_1,\dots,a_k) \iff D(a_1,\dots,a_k) \text{ dominates all } l\text{-cliques of } \mathcal{G}'.$$

This completes our proof that $\mathrm{DC}^{\mathrm{dir}}$ is A[2]-hard.

For undirected graphs, we essentially use the same construction. The only problem is that the undirected graph underlying $\mathcal{G}'$ may contain additional cliques of size l that we have not taken into account so far. Specifically, such cliques (let us call them *bad cliques*) may be generated by the edges introduced in (9) and (11). Therefore, each bad clique contains a vertex $(u,a)\in\bigcup_{u=1}^{k}\{u\}\times G$ or a vertex $(u,v,a,b)\in\bigcup_{1\leqslant u<v\leqslant k}\{(u,v)\}\times G^2$. To make sure that all bad cliques are dominated we add to the undirected graph underlying $\mathcal{G}'$ a clique D of new vertices $d_1,\dots,d_l$ and a further vertex d together with edges from d to d_1 and to all elements of

$$\bigcup_{u=1}^{k}\{u\}\times G \;\cup\; \bigcup_{1\leqslant u<v\leqslant k}\{(u,v)\}\times G^2.$$

Then every set of vertices of the graph dominating all l-cliques must contain d. Now the proof goes through for the new parameter $1+k+\binom{k}{2}$. □

3.1. Logical descriptions

In this section, we give logical descriptions of the classes introduced above. Let us first review Theorem 27 in the light of slicewise definability. Observe that the problem p-MC(ORD[τ], L[τ]), for a logic L, is in a sense the "canonical" slicewise L-definable problem; it is slicewise definable via a function δ that simply decodes $[\varphi]$ back to φ. Actually, it is not essential here to have ordered structures, for a vocabulary τ that, without loss of generality, does not contain any of the symbols $\leqslant$, min, max, S we can simply identify p-MC(L[τ]) with the problem

$$\{(\mathcal{A},[\varphi]) \mid \mathcal{A}\in \mathrm{ORD}[\tau], \varphi\in \mathrm{L}[\tau], \mathcal{A}\models\varphi\}\subseteq \mathrm{ORD}[\tau]\times\mathbb{N}.$$

So Theorem 27 at least shows that for the classes A[t], for $t\geqslant 1$, and AW[$*$] we have complete problems that are slicewise definable in nice and simple logics. For the classes W[t], for $t\geqslant 1$, the situation is similar, though slightly more complicated. But of course these are not yet capturing results in the style of Subsection 2.2.

Proposition 21 tells us that if $\mathrm{A}[t]=\bigcup_{s\geqslant 1}$ slicewise-L_s or $\mathrm{W}[t]=\bigcup_{s\geqslant 1}$ slicewise-L_s then $\bigcup_{s\geqslant 1}\mathrm{L}_s=\mathrm{LFP}$. So again, we have to look for suitable "filtrations" of LFP. Recall that LFP^s denotes the class of formulas of least fixed-point logic of the form $[\mathrm{LFP}_{\bar{x},X}\,\psi]\,\bar{t}$, where $\psi\in\mathrm{FO}^s$ and where $\bar{t}$ has length at most s. Let $\mathrm{BOOL}(\mathrm{LFP}^s)$ be the class of Boolean combinations φ of formulas in LFP^s. Note that φ itself may contain more than s variables. For $t\geqslant 1$, we let Σ_t-$\mathrm{BOOL}(\mathrm{LFP}^s)$ be the class of all first-order formulas of the form

$$\exists x_{11}\ldots\exists x_{1\ell_1}\,\forall x_{21}\ldots\forall x_{2\ell_2}\,\exists x_{31}\ldots\exists x_{3\ell_3}\;\ldots\;Qx_{t1}\ldots Qx_{t\ell_t}\;\chi, \tag{12}$$

where $\chi\in\mathrm{BOOL}(\mathrm{LFP}^s)$ and $Q=\exists$ if t is odd, $Q=\forall$ if t is even. For $u\geqslant 1$, we let $\Sigma_{t,u}$-$\mathrm{BOOL}(\mathrm{LFP}^s)$ be the class of all Σ_t-$\mathrm{BOOL}(\mathrm{LFP}^s)$ formulas of the form (12) where $\ell_2,\ell_3,\ldots,\ell_t\leqslant u$. Finally, we let $\mathrm{FO}(\mathrm{LFP}^s)$ be the closure of LFP^s under Boolean combinations and first-order existential and universal quantification.

Theorem 29.
(1) *For all* $t\geqslant 1$, *we have* $\mathrm{W}[t]=\bigcup_{u\geqslant 1}\bigcup_{s\geqslant 1}$ slicewise-$\Sigma_{t,u}$-$\mathrm{BOOL}(\mathrm{LFP}^s)$.
(2) *For all* $t\geqslant 1$, *we have* $\mathrm{A}[t]=\bigcup_{s\geqslant 1}$ slicewise-Σ_t-$\mathrm{BOOL}(\mathrm{LFP}^s)$.
(3) $\mathrm{AW}[*]=\bigcup_{s\geqslant 1}$ slicewise-$\mathrm{FO}(\mathrm{LFP}^s)$.

Proof. The following well-known Lemma can easily be proved using part (1) of Lemma 24. Recall that for a structure $\mathcal{A}$ and a formula $\varphi(x_1,\ldots,x_k)$, $\varphi(\mathcal{A})$ denotes the set of all tuples $(a_1,\ldots,a_k)\in A^k$ such that $\mathcal{A}\models\varphi(a_1,\ldots,a_k)$. □

Lemma 30. *There is an algorithm that for every* $\mathcal{A}\in\mathrm{ORD}[\tau]$ *and* $\varphi\in\mathrm{LFP}^s[\tau^{\mathrm{ord}}]$ *computes the set* $\varphi(\mathcal{A})$ *in time* $O(|\varphi|\cdot|\mathcal{A}|^{2s})$.

We need another Lemma, which is a strengthening of some of the statements of Theorem 27. The *arity* of a vocabulary τ is the maximum of the arities of the relation symbols in τ, or 1, if τ does not contain any relation symbols. For a class Φ of formulas and an $s \geqslant 1$ we let $\Phi[s]$ denote the class of all formulas in Φ whose vocabulary is at most s-ary.

Lemma 31 [15]**.**
(1) *For all* $s, t, u \geqslant 1$ *we have* p-MC$(\Sigma_{t,u}[s]) \in$ W$[t]$.
(2) *For all* $s, t \geqslant 1$ *we have* p-MC$(\Sigma_t[s]) \in$ A$[t]$.
(3) p-MC(FO$[s]) \in$ AW$[*]$.

Statement (1) of the Lemma does not appear in [15], but can easily be derived with the techniques used there (specifically, in the proof of Lemma 3.2).

Proof (of Theorem 29). We just prove statement (1). The proofs of (2) and (3) are very similar (and slightly simpler). Let $s, t, u \geqslant 1$ and $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$.

First, assume that P is slicewise $\Sigma_{t,u}$-BOOL(LFP^s)-definable via $\delta : \mathbb{N} \to \Sigma_{t,u}$-BOOL(LFP^s)$[\tau^{\mathrm{ord}}]$. Let r be the arity of τ^{ord} and $s' = \max\{r, s\}$. We shall prove that $P \leqslant^{\mathrm{FPT}}$ p-MC$(\Sigma_{t,u}[s'])$ and thus by Lemma 31(1), that $P \in$ W$[t]$.

Let $\mathcal{A} \in \mathrm{ORD}[\tau]$ and $k \in \mathbb{N}$. Suppose that

$$\delta(k) = \exists \bar{x}_1 \forall \bar{y}_2 \ldots Q \bar{y}_t \, \chi$$

with $\chi \in$ BOOL(LFP^s). Then χ is a Boolean combination of the LFP^s-formulas $\varphi_1(\bar{x}_1), \ldots, \varphi_m(\bar{x}_m)$, where $\bar{x}_i$ has length s_i for some $s_i \leqslant s$. For $i = 1, \ldots, m$ let R_i be a new s_i-ary relation symbol. Let $\chi(R_1/\varphi_1, \ldots, R_m/\varphi_m)$ be the FO$[\tau \cup \{R_1, \ldots, R_m\}]$-formula obtained from χ by replacing $\varphi_i(\bar{x}_i)$ by $R_i(\bar{x}_i)$ for $i = 1, \ldots, m$. Then, $\delta(k)' := \exists \bar{x}_1 \forall \bar{y}_2 \ldots Q \bar{y}_t \chi(R_1/\varphi_1, \ldots, R_m/\varphi_m)$ is a $\Sigma_{t,u}[s']$-formula. Furthermore, let $\mathcal{A}'$ be the $\tau^{\mathrm{ord}} \cup \{R_1, \ldots, R_m\}$-expansion of $\mathcal{A}$ obtained by setting $R_i^{\mathcal{A}'} := \varphi_i(\mathcal{A})$ for $i = 1, \ldots, m$. We claim that the mapping $(\mathcal{A}, k) \mapsto (\mathcal{A}', [\delta(k)'])$ is an FPT-reduction from P to p-MC$(\Sigma_{t,u}[s'])$. Our construction immediately shows that

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \delta(k) \iff \mathcal{A}' \models \delta(k)' \iff (\mathcal{A}, [\delta(k)']) \in \text{p-MC}(\Sigma_{t,u}[s']).$$

Moreover, the function $k \mapsto [\delta(k)']$ is computable. It remains to show that the function $(\mathcal{A}, k) \mapsto \mathcal{A}'$ is computable in time $f(k) \cdot |\mathcal{A}|^c$ for some computable function f and constant c. But this follows from Lemma 30 and the fact that the mapping δ is computable.

For the converse direction, assume that $P \subseteq \mathrm{ORD}[\tau] \times \mathbb{N}$ is in W$[t]$. Let E be a binary relation symbol. By Theorem 27(1) there is a $u \geqslant 1$ such that $P \leqslant^{\mathrm{FPT}}$ p-MC$(\Sigma_{t,u}[\{E\}])$. For notational simplicity, we assume $u = 1$.

Thus, there are computable functions $f, g : \mathbb{N} \to \mathbb{N}$, a constant c, and an algorithm $\mathbb{A}$ that with every $(\mathcal{A}, k) \in \mathrm{ORD}[\tau] \times \mathbb{N}$ associates an $\{E\}$-structure $\mathcal{G} = \mathcal{G}(\mathcal{A}, k)$ and a sentence $\varphi \in \Sigma_{t,1}$, $\varphi = \varphi(\mathcal{A}, k)$, in time at most $f(k) \cdot |\mathcal{A}|^c$ such that

$$(\mathcal{A}, k) \in P \iff \mathcal{G} \models \varphi \tag{13}$$

and $[\varphi] \leqslant g(k)$. Without loss of generality we can assume $f(k) \geqslant k$ for all k.

For $k \in \mathbb{N}$ let $\mathbb{A}_k$ be an algorithm that, on input $\mathcal{A} \in \mathrm{ORD}[\tau]$, simulates $\mathbb{A}$ on input $(\mathcal{A}, k)$. Thus $\mathbb{A}_k$ is an algorithm that accepts the kth slice of P. For $\mathcal{A} \in \mathrm{ORD}[\tau]$ with $|\mathcal{A}| > f(k)$, the algorithm $\mathbb{A}_k$

works in time $|\mathcal{A}|^{c+1}$. Thus by (a slight extension of) Theorem 17(3), there is an $s \geqslant 1$ (not depending on k) such that the algorithm $\mathbb{A}_k$ can be simulated in $\mathcal{A}$ by LFP^s-formulas. This means two things. First, it means that there is a pair $\Psi = (\psi_{\text{uni}}(\bar{x}), \psi_E(\bar{x}, \bar{y}))$ of formulas in LFP$^s[\tau^{\text{ord}}]$ such that for all $\mathcal{A} \in \text{ORD}[\tau]$ with $|\mathcal{A}| > f(k)$ the $\{E\}$-structure

$$\mathcal{A}^\Psi := \Big(\psi_{\text{uni}}(\mathcal{A}), \psi_E(\mathcal{A}) \cap \big(\psi_{\text{uni}}(\mathcal{A})\big)^2\Big)$$

is isomorphic to the structure $\mathcal{G}(\mathcal{A}, k)$. And second, it means that there is a formula $\xi(x) \in \text{LFP}^s[\tau^{\text{ord}}]$ such that for all $\mathcal{A} \in \text{ORD}[\tau]$ with $|\mathcal{A}| > f(k), g(k)$ and all m we have

$$\mathcal{A} \models \xi(S^m(\min)) \iff m = [\varphi(\mathcal{A}, k)]. \tag{14}$$

Moreover, the length ℓ of $\bar{x}$ and $\bar{y}$ in $\psi_{\text{uni}}(\bar{x})$, $\psi_E(\bar{x}, \bar{y})$ does not depend on k. We can assume that there are distinct variables $x_1, \ldots, x_{p(k)}, y_2, \ldots, y_t$ such that all the sentences of the form $\varphi(\mathcal{A}, k)$ with $\mathcal{A} \in \text{ORD}[\tau]$ have the form

$$\exists z_1 \ldots \exists z_q \forall y_2 \ldots Q y_t \rho \tag{15}$$

for some quantifier-free ρ and with $z_1, \ldots, z_q \in \{x_1, \ldots, x_{p(k)}\}$. For all $i \leqslant g(k)$, if the sentence χ with $[\chi] = i$ is of the form (15), then let $\chi_i = \chi$, otherwise let χ_i be undefined. If χ_i is defined, let ρ_i be the corresponding ρ in (15), otherwise let $\rho_i = (x_1 \neq x_1)$. Choose a sequence $\bar{x}_1, \ldots, \bar{x}_{p(k)}, \bar{y}_2, \ldots, \bar{y}_t$ of disjoint ℓ-tuples of variables (recall that ℓ denotes the length of $\bar{x}$ and $\bar{y}$ in $\psi_{\text{uni}}(\bar{x})$, $\psi_E(\bar{x}, \bar{y})$). Then every ρ_j has a natural translation into a τ^{ord}-formula ρ_j^Ψ obtained by replacing atomic formulas of the form Euv and $u = v$ by $\psi_E(\bar{u}, \bar{v})$ and $\bigwedge_{i<\ell} u_i = v_i$, respectively. Writing $\exists \bar{x} \in \psi_{\text{uni}}\, \varphi$ instead of $\exists \bar{x}(\psi_{\text{uni}}(\bar{x}) \wedge \varphi)$ and $\forall \bar{x} \in \psi_{\text{uni}}\, \varphi$ instead of $\forall \bar{x}(\psi_{\text{uni}}(\bar{x}) \to \varphi)$, we let

$$\chi_i^\Psi = \exists \bar{x}_1 \in \psi_{\text{uni}} \ldots \exists \bar{x}_{p(k)} \in \psi_{\text{uni}} \forall \bar{y}_2 \in \psi_{\text{uni}} \exists \bar{y}_3 \in \psi_{\text{uni}} \ldots Q \bar{y}_t \in \psi_{\text{uni}}\, \rho_i^\Psi.$$

Then for all $\mathcal{A} \in \text{ORD}[\tau]$ with $|\mathcal{A}| > f(k)$ we have

$$\mathcal{A} \models \chi_i^\Psi \iff \mathcal{A}^\Psi \models \chi_i. \tag{16}$$

Next, we shall define a formula $\delta_k \in \Sigma_{t,\ell}\text{-BOOL}(\text{LFP}^s)$ with the property that

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \delta_k. \tag{17}$$

Furthermore, we shall prove that the mapping $k \mapsto \delta_k$ is computable, which completes our proof that P is slicewise $\Sigma_{t,\ell}\text{-BOOL}(\text{LFP}^s)$-definable

We let

$$\delta_k' = \bigvee_{i \leqslant g(k)} \big(\xi(S^i(\min)) \wedge \chi_i^\Psi\big).$$

Then by (13), (14), (16), and the fact that $\mathcal{G}(\mathcal{A}, k) \cong \mathcal{A}^\Psi$, for all $\mathcal{A}$ with $|\mathcal{A}| > f(k), g(k)$ we have

$$(\mathcal{A}, k) \in P \iff \mathcal{A} \models \delta_k'.$$

By standard means we can transform δ_k' into an equivalent formula

$$\delta_k'' = \exists \bar{x}_1 \ldots \exists \bar{x}_{p(k)} \forall \bar{y}_2 \ldots Q \bar{y}_t\ \zeta_k,$$

where ζ_k is a Boolean combination of the formulas $\xi(S^i(\min))$, for $1 \leqslant i \leqslant g(k)$, $\psi_{\text{uni}}(\bar{x}_i)$, for $1 \leqslant i \leqslant g(k)$, $\psi_{\text{uni}}(\bar{u}_i)$, for $2 \leqslant i \leqslant t$, and ρ_i^Ψ, for $1 \leqslant i \leqslant g(k)$. Then $\zeta_k \in \text{BOOL}(\text{LFP}^s)$.

We let

$$\delta_k''' = \bigvee_{\substack{(\mathcal{A},k)\in P\\ |\mathcal{A}|\leqslant \max\{f(k),g(k)\}}} \varphi_{\mathcal{A}} \quad \vee \Big(S^{f(k)-1} \neq \max \wedge S^{g(k)-1} \neq \max \wedge \delta_k''\Big).$$

Then for all $\mathcal{A}$ we have

$$(\mathcal{A},k)\in P \iff \mathcal{A}\models \delta_k'''.$$

Now we can easily transform δ_k''' into an equivalent formula

$$\delta_k = \exists \bar{x}_1 \ldots \exists \bar{x}_{p(k)} \forall \bar{y}_2 \ldots Q\bar{y}_t\ \zeta_k',$$

where ζ_k' is a Boolean combination of ζ_k and the formulas $\varphi_{\mathcal{A}}$, for τ-structures $\mathcal{A}$ with $(\mathcal{A},k)\in P$ and $|\mathcal{A}|\leqslant \max\{f(k),g(k)\}$, $S^{f(k)-1}\neq \max$, and $S^{g(k)-1}\neq \max$. Then $\delta_k \in \Sigma_{t,\ell}$-BOOL(LFP^s). Since δ_k is equivalent to δ_k''' (17) holds.

Finally, we have to show that the mapping $k\mapsto \delta_k$ is computable. To do this, we first note that for every k we can compute an algorithm $\mathbb{A}_k$ that accepts the kth slice of P in time $f(k)\cdot|\mathcal{A}|^c$ (because we know $\mathbb{A}$). Then we use the fact that the proof of Theorem 17(3) actually gives a computable translation of polynomial time algorithms into LFP-formulas. Thus both Ψ and ξ are computable from k. Since $g(k)$ is computable, we can compute the set of all FO-formulas χ with $[\chi]\leqslant g(k)$. Since f, g, and the syntactical operations that were used to obtain a formula δ_k of the right form are also computable, the formula δ_k can be computed from k. □

Remark 32. The results remain true if we replace LFP^s by LFP$_s$.

The following simple observation together with the above descriptive characterisations yields model-theoretic versions of open complexity-theoretic problems.

Proposition 33. *Let* Q *and* Q′ *be parametric classes with* $\mathrm{Q}=\bigcup_{s\geqslant 1}$ *slicewise-*L_s *and* $\mathrm{Q}'=\bigcup_{s\geqslant 1}$ *slicewise-*L'_s. *Moreover, assume that* $\mathrm{L}'_s[\tau^{\mathrm{ord}}]$ *is a computable set for every vocabulary* τ *and every* $s\geqslant 1$. *Then*

$$\mathrm{Q}'\subseteq \mathrm{Q} \iff \forall\tau\,\forall s\geqslant 1\,\exists r\geqslant 1:\ \mathrm{L}'_s[\tau^{\mathrm{ord}}]\subseteq \mathrm{L}_r[\tau^{\mathrm{ord}}].^{8}$$

Proof. The implication from right to left being trivial, we turn to the other direction: Fix a vocabulary τ and $s\geqslant 1$. Choose a computable and surjective function $\delta:\mathbb{N}\to \mathrm{L}'_s[\tau^{\mathrm{ord}}]$. Define the problem $P\subseteq \mathrm{ORD}[\tau]\times\mathbb{N}$ by

$$(\mathcal{A},k)\in P \iff \mathcal{A}\models \delta(k).$$

Then, $P\in \mathrm{Q}'$ and hence by assumption, $P\in \mathrm{Q}$. Therefore, P is slicewise-L_r definable for some $r\geqslant 1$. But then $\mathrm{L}'_s[\tau^{\mathrm{ord}}]\subseteq \mathrm{L}_r[\tau^{\mathrm{ord}}]$. □

[8] For logics L and L′, $\mathrm{L}'[\tau^{\mathrm{ord}}]\subseteq \mathrm{L}[\tau^{\mathrm{ord}}]$ means that every $\mathrm{L}'[\tau^{\mathrm{ord}}]$ is equivalent to an $\mathrm{L}[\tau^{\mathrm{ord}}]$-sentence on ordered structures.

In particular, we get

Corollary 34.

(1) $\mathrm{FPT} = W[1] \iff \forall\tau\,\forall s \geqslant 1\,\exists r \geqslant 1 :\ \Sigma_1\text{-BOOL}(\mathrm{LFP}^s)[\tau^{\mathrm{ord}}] \subseteq \mathrm{LFP}^r[\tau^{\mathrm{ord}}]$.

(2) $W[t] = A[t] \iff \forall\tau\,\forall s \geqslant 1\,\exists u, r \geqslant 1 :\ \Sigma_t\text{-BOOL}(\mathrm{LFP}^s)[\tau^{\mathrm{ord}}] \subseteq \Sigma_{t,u}\text{-BOOL}(\mathrm{LFP}^r)[\tau^{\mathrm{ord}}]$.

4. Conclusions

We have approached parameterized complexity theory from the perspective of classical complexity theory and descriptive complexity theory. By studying the parameterized analogues para-K of classical classes K and transferring results from the classical world to the parameterized world, we have gained a good understanding of a large family of parameterized complexity classes, among them the class FPT of fixed-parameter tractable problems. Our descriptive complexity theoretic approach took us even further; we gave natural logical descriptions of most of the important parameterized complexity classes.

In this paper, we were mainly concerned with the overall structure of parameterized complexity classes. Some of the particular classes para-K deserve further study. In particular, we would like to mention the classes para-LOGSPACE and para-NLOGSPACE here. In Example 6, we have proved a quite general result showing that many interesting properties of graphs of bounded degree are in para-LOGSPACE. It would be interesting to see natural parameterized problems that are fixed-parameter tractable, but can be proved to be not in para-LOGSPACE (under the complexity theoretic assumption that FPT $\neq$ para-LOGSPACE). The interesting twist in this problem is that we cannot simply prove such problems to be FPT-complete under PL-reductions, because by Proposition 15, FPT-complete problems are in some sense trivial from a parameterized point of view.

References

[1] K.A. Abrahamson, R.G. Downey, M.R. Fellows, Fixed-parameter tractability and completeness IV: on completeness for W[P] and PSPACE analogs, Ann. Pure Appl. Logic 73 (1995) 235–276.

[2] M. Alekhnovich, A. Razborov, Resolution is not automatizable unless W[P] is tractable, in: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science, 2001, pp. 210–219.

[3] H.L. Bodlaender, R.G. Downey, M.R. Fellows, M.T. Hallett, H.T. Wareham, Parameterized complexity analysis in computational biology, Comput. Appl. Biosci. 11 (1995) 49–57.

[4] J.R. Büchi, Weak second-order arithmetic and finite automata, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik 6 (1960) 66–92.

[5] L. Cai, J. Chen, R.G. Downey, M.R. Fellows, Advice classes of parameterized tractability, Ann. Pure Appl. Logic 84 (1997) 119–138.

[6] Y. Chen, J. Flum, Machine characterizations of the classes of the W-hierarchy, in: Proceedings of the 17th International Workshop on Computer Science Logic, Lecture Notes in Computer Science, Springer-Verlag, 2003, to appear.

[7] Y. Chen, J. Flum, M. Grohe, Bounded nondeterminism and alternation in parameterized complexity theory, in: Proceedings of the 18th IEEE Conference on Computational Complexity, pp. 13–29, 2003.

[8] R.G. Downey, M.R. Fellows, Fixed-parameter tractability and completeness I: basic results, SIAM J. Comput. 24 (1995) 873–921.

[9] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, Berlin, 1999.

[10] R.G. Downey, M.R. Fellows, K. Regan, Descriptive complexity and the W-hierarchy, in: P. Beame, S. Buss (Eds.), Proof Complexity and Feasible Arithmetic, AMS-DIMACS Volume Series, vol. 39, AMS, Providence, RI, 1998, pp. 119–134.
[11] R.G. Downey, M.R. Fellows, K. Taylor, The parameterized complexity of relational database queries and an improved characterization of $W[1]$, in: D.S. Bridges, C. Calude, P. Gibbons, S. Reeves, I.H. Witten (Eds.), Combinatorics, Complexity, and Logic—Proceedings of DMTCS '96, Springer, Berlin, 1996, pp. 194–213.
[12] H.-D. Ebbinghaus, J. Flum, Finite Model Theory, second ed., Springer, Berlin, 1999.
[13] H.-D. Ebbinghaus, J. Flum, W. Thomas, Mathematical Logic, second ed., Springer, Berlin, 1994.
[14] R. Fagin, Generalized first-order spectra and polynomial-time recognizable sets, in: R.M. Karp (Ed.), Complexity of Computation, SIAM-AMS Proceedings, vol. 7, 1974, pp. 43–73.
[15] J. Flum, M. Grohe, Fixed-parameter tractability, definability, and model checking, SIAM J. Comput. 31 (1) (2001) 113–145.
[16] H. Gaifman, On local and non-local properties, in: J. Stern (Ed.), Proceedings of the Herbrand Symposium, Logic Colloquium '81, North Holland, 1982, pp. 105–135.
[17] G. Gottlob, N. Leone, M. Sideri, Fixed-parameter complexity in AI and nonmonotonic reasoning, in: M. Gelfond, N. Leone, G. Pfeifer (Eds.), Logic Programming and Nonmonotonic Reasoning, 5th International Conference, LPNMR'99, Lecture Notes in Computer Science, vol. 1730, Springer, Berlin, 1999, pp. 1–18.
[18] M. Grohe, Generalized model-checking problems for first-order logic, in: H. Reichel, A. Ferreira (Eds.), Proceedings of the 18th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Computer Science, vol. 2010, Springer, Berlin, 2001, pp. 12–26.
[19] M. Grohe, The parameterized complexity of database queries, in: Proceedings of the 20th ACM Symposium on Principles of Database Systems, 2001, pp. 82–92.
[20] M. Grohe, T. Schwentick, L. Segoufin, When is the evaluation of conjunctive queries tractable, in: Proceedings of the 33rd ACM Symposium on Theory of Computing, 2001, pp. 657–666.
[21] N. Immerman, Relational queries computable in polynomial time, Inform. Control 68 (1986) 86–104.
[22] N. Immerman, Languages that capture complexity classes, SIAM J. Comput. 16 (1987) 760–778.
[23] N. Immerman, Descriptive Complexity, Springer, Berlin, 1999.
[24] C.H. Papadimitriou, M. Yannakakis, On the complexity of database queries, in: Proceedings of the 17th ACM Symposium on Principles of Database Systems, 1997, pp. 12–19.
[25] C.H. Papadimitriou, Computational Complexity, Addison–Wesley, 1994.
[26] U. Stege, Resolving conflicts in problems from computational biology. PhD thesis, ETH Zuerich, 2000. PhD thesis No.13364.
[27] L.J. Stockmeyer, The complexity of decision problems in automata theory. PhD thesis, Department of Electrical Engineering, MIT, 1974.
[28] M.Y. Vardi, The complexity of relational query languages, in: Proceedings of the 14th ACM Symposium on Theory of Computing, 1982, pp. 137–146.